INFORMATION AND COMPUTATION 122, 1–29 (1995)

Observable Behaviors and Equivalences of Logic Programs

MAURIZIO GABBRIELLI

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy; and CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
E-mail: gabbri@di.unipi.it and gabbri@cwi.nl

GIORGIO LEVI AND MARIA CHIARA MEO

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy
E-mail: {levi,meo}@di.unipi.it

We first introduce a general semantic scheme for logic programs which provides a uniform framework for defining different compositional semantics parametrically with respect to a given notion of observability. The equivalence of the operational (top-down) and fixpoint (bottom-up) construction of the semantics is ensured by the scheme (provided a congruence property is verified). We then define several observational equivalences on logic programs and investigate how they are related. The equivalences are based on various observables (successful derivations, computed answers, partial computed answers and call patterns) and on a notion of program composition. For each observational equivalence we study the relation with a suitable formal semantics by investigating correctness and full abstraction properties. All the semantics we consider are obtained as instances of the general scheme.

1. INTRODUCTION

According to [36], *the* semantics of positive logic programs is defined in model-theoretic terms (the least Herbrand model) and can be *computed* by a top-down construction (the success set) and by a bottom-up construction (the least fixpoint of the immediate consequences operator). Is the above classical and elegant result satisfactory? The answer can be found if we first consider a more basic question. What is a semantics used for? The first *application* of any semantics is to aid in the understanding of the meaning of programs. This is the basis for any other application such as program transformation and program analysis. *Program understanding* is based on our ability to detect when two programs cannot be distinguished by looking at their behaviors. Therefore defining an equivalence on (logic) programs $\approx$ and on a formal semantics $\mathscr{S}(P)$ are two strongly related tasks. In general a semantics $\mathscr{S}(P)$ for logic programs is *correct* wrt $\approx$, if $\mathscr{S}(P_1) = \mathscr{S}(P_2)$ implies $P_1 \approx P_2$. $\mathscr{S}(P)$ is *fully abstract* wrt $\approx$, if the converse holds, also; i.e., $\mathscr{S}(P_1) = \mathscr{S}(P_2)$ iff $P_1 \approx P_2$. While full abstraction is known to be a desirable property for any semantics, correctness is a must. The question on the adequacy of van Emden and Kowalski's semantics can then be rephrased as follows. Is that semantics correct wrt a "natural" notion of program equivalence? And this in turn raises the problem of choosing a suitable program equivalence.

Equivalences can be defined by using logical arguments only. For example, one can use model-theoretic properties, such as the set of logical consequences or the least Herbrand model, and proof-theoretic properties, such as the set of derivable atoms. However, this is not completely satisfactory since when we try to understand the meaning of programs, or analyze and transform programs, we need equivalences (and semantics) which capture the "observational" behavior of programs. The pure logical reading of programs and equivalences based on it fail to capture interesting properties of computations since there exist several observable properties which cannot be expressed in logical terms. For example, logically equivalent programs can compute different answer substitutions for the same goal (Example 2.9) and vice versa (of course, programs which compute the same answers need not be logically equivalent).

Therefore, the equivalences we are concerned with are *observational equivalences*, i.e., equivalences $\approx_x$ on programs which are induced by a given notion of observable property x of program computations. For example, *computed answer substitution* (ca) is the most natural observable since it is usually considered the result of a program execution. The equivalence $\approx_{ca}$ induced by such an observable on programs can be defined as follows: $P_1 \approx_{ca} P_2$ iff, for every goal G, G has the same computed answers both in P_1 and in P_2. A more abstract observable like *successful derivations* would fail in capturing the essence of logic programming, even if it is adequate to the case of first-order theorem proving, where there is nothing to be returned as a computation result. It is worth noting also that *abstract interpretation* can be understood and formalized in terms of the choice of a suitable observable (for example groundness of goal variables in the derivation).

0890-5401/95 $12.00

For each observational equivalence $\approx_x$ we want to find a *correct* semantics and, possibly, one fully abstract wrt $\approx_x$. In particular, any "reference semantics" for logic programs should be correct wrt the equivalence $\approx_{ca}$ since, as previously discussed, computed answers capture the essence of computing with logic programs. However, as first shown in [17], this is not the case for the classical van Emden and Kowalski semantics which is correct wrt successful derivations only. The need for a formal semantics correct wrt $\approx_{ca}$ gave rise to several new definitions and was recognized especially in the case of semantics-based abstract interpretation [32, 3, 11] and program transformation [25, 6], where also less abstract observables (such as partial computed answers) sometimes have to be modeled [11].

In addition to the problem of modeling observational equivalences, there exists an important property which does not hold in the least Herbrand model semantics, i.e. *compositionality* wrt the union of programs. A semantics is compositional wrt a (syntactic) program composition operator $\circ$ when the semantics of the compound construct $C_1 \circ C_2$ can be defined by (semantically) composing the semantics of the constituents C_1 and C_2. In the case of logic programs the least Herbrand model semantics is not compositional wrt the *union* of programs (i.e., it is not OR-compositional). If the observable is successful derivations, OR-compositionality can be understood in logical terms since the set of all the (Herbrand) models is OR-compositional [30] (and correct wrt successful derivations). The only OR-compositional semantics correct wrt computed answers are described in [24, 9, 8], while all the other OR-compositional semantics [23, 26, 30, 31, 33] are only correct wrt successful derivations. Clearly, compositionality wrt program composition operators is a desirable property since it allows one to define in a modular way and incrementally the semantics of structured programs. For example, a semantics compositional wrt a generalized composition operator has been introduced in [5] to characterize logic programs structured by inheritance mechanisms. Moreover, compositionality is the base on which to develop semantics based tools for the modular analysis and transformation of programs.

Over the last few years we have developed a general approach to the semantics [7, 21] whose aim was modeling the observable behaviors (possibly in a compositional way) for a variety of logic languages, ranging from positive logic programs [17, 9, 8] to constraint logic programs [20, 19] and general logic programs [35].

Our approach is based on the idea of choosing (equivalence classes of) sets of clauses as semantic domains. Denotations (that we call π-interpretations) are not interpretations in the conventional mathematical logic sense and can be computed both by a top-down construction (a success set) and by a bottom-up construction (the least fixpoint of suitable continuous immediate consequence operators on π-interpretations).

The s-semantics [17] was the first (non-compositional) semantics correct wrt computed answers and which used sets of unit clauses as its domain. Gaifman and Shapiro, following a proof-theoretic approach, introduced the idea of using sets of non-unit clauses to achieve OR-compositionality [23]. They then defined (using a different semantic domain) a fully abstract OR-compositional semantics modeling computed answers [24]. The Ω-semantics [9, 8] is the real compositional generalization of the s-semantics and uses non-unit clauses as domain.

In this paper, following the approach of [21], we first define a semantics scheme which generalizes the Ω-semantics and provides a uniform framework for defining different compositional semantics for logic programs parametrically wrt a given notion of observability. The scheme allows both top-down (operational) and bottom-up (fixpoint) definitions and ensures the equivalence of the two characterizations, provided a congruence property is verified. Moreover, the scheme allows one to treat in a uniform way both "concrete" semantics and "abstract" semantics such as those arising in semantics based program analysis. Indeed, an abstract semantics is simply obtained by considering an "abstract" notion of observable (e.g., groundness, term rigidity, variable sharing, etc.). The correctness of an abstract semantics wrt a concrete one is usually stated by using a Galois connection which relates the two semantics [12]. It is worth noting that in our framework, since all the semantics are instances of the same scheme, their relations can be uniformly understood in terms of abstractions. Therefore, the relation between different semantics (not necessarily a concrete and an abstract one) could be expressed by using the formal tools of abstract interpretation.

We then consider various program equivalence notions and investigate how they are related to study their discriminating power. A systematic comparison of various program equivalences has already been given in [30], which, however, considers equivalences based either on logical properties or on the semantics in [36]. We have already argued that all these equivalences are too weak to correctly characterize computed answers. For example, programs $P_1 = \{p(X).\ p(a).\}$ and $P_2 = \{p(X).\}$ are identified by all the equivalences considered in [30], and the same is true for programs $Q_1 = \{q(X) : -q(X),\ q(X).\}$ and $Q_2 = \{q(X) : -q(X).\}$. However, programs P_1 and P_2 have different computed answers (see Example 2.9) and, if we consider the OR-composition of programs, Q_1 and Q_2 are different too (see Example 4.14). Some equivalences which consider OR-composition have already been studied in [23] by considering successful derivations only, and in [24] by considering also computed answers. However, programs $R_1 = \{p(a) : -q(a).\ q(a).\}$ and $R_2 = \{p(a).\ q(a).\}$ are identified by all the equivalences considered in [23, 24], while if we consider (correct) call patterns they can be distinguished (see Example 2.9).

We extend the study of [30] and [23, 24] by introducing new equivalences (which are able to capture differences such as those of the previous programs) and by analyzing their relationships. The new equivalences are obtained both by considering new observables (such as *partial computed answers* and *call patterns*) and by considering non-standard T_P operators. For each observational equivalence we study how it is related to a suitable formal semantics by investigating correctness and full abstraction properties. For example, the s-semantics [17] is correct and fully abstract wrt "computed answers" (and therefore allows to distinguish programs P_1 and P_2), while the Ω-semantics [9, 8, 21] is also OR-compositional (and therefore distinguishes programs Q_1 and Q_2). For partial answers we obtain a fully abstract semantics and for call patterns a correct one (which allows us to distinguish programs R_1 and R_2). Having related each observational equivalence to a suitable fixpoint semantics and therefore to an immediate consequence operator, we then study the equivalences induced by these operators and their relations to the observational ones. All the semantics that we consider are obtained as specific instances of our general scheme.

The remainder of the paper is organized as follows. The next subsection contains notation and preliminaries. Section 2 introduces the general notions of observable equivalences and their relations to formal semantics. Moreover, it shows the composition operator and the specific observables we will use in the following. Section 3 introduces our general semantic framework and gives the general result of equivalence between the top-down and the bottom-up constructions. In Section 4 we consider equivalences based on computed answers and successful derivations, when program composition is the union of predicate disjoint programs (i.e., programs not sharing predicate names). In particular, some results of [30] are extended to the non-ground" case. The more general case of program union is considered in Section 5 where a full abstraction result is given for a non-ground functional semantics. Section 6 introduces some new observables (partial answers and call patterns) and shows specific instances of the general scheme which allow one to obtain correct (in one case fully abstract) semantics for them. Section 7 is devoted to some concluding remarks. All of the proofs which are not shown in these sections are contained in the appendix. A short version of this paper appeared in [22].

1.1. *Preliminaries*

The reader is assumed to be familiar with the terminology and the main results on the semantics of logic programs. In this subsection we introduce some notations we will use in the following and, for the reader's convenience, we recall some basic notions. We refer to the reader [28, 1] for further details on the logic programming theory.

Throughout the paper we will assume programs and goals are defined on a first order language given by a signature Σ consisting of a denumerable set F of *function symbols*, a finite set Π of *predicate symbols*, and a denumerable set V of *variable symbols*. Let T be the set of terms built on F and V. Variable-free terms are called *ground*. A substitution is a mapping $\vartheta: V \to T$ such that the set $D(\vartheta) = \{X \mid \vartheta(X) \neq X\}$ (*domain* of ϑ) is finite. If $W \subseteq V$, we denote by $\vartheta_{|W}$ the *restriction* of ϑ to the variables in W, i.e., $\vartheta_{|W}(Y) = Y$ for $Y \notin W$. ε denotes the empty substitution. The *composition* $\vartheta\sigma$ of the substitutions ϑ and σ is defined as the functional composition. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \varepsilon$. The pre-ordering $\leqslant$ (more general than) on substitutions is such that $\vartheta \leqslant \sigma$ iff there exists ϑ' such that $\vartheta\vartheta' = \sigma$. The result of the application of the substitution ϑ to a term t is an *instance* of t and is denoted by $t\vartheta$. We define $t \leqslant t'$ (t is more general than t') iff there exists ϑ such that $t\vartheta = t'$. The relation $\leqslant$ is a preorder (called subsumption) and by $\equiv_v$ we denote the associated equivalence relation (*variance*). A substitution ϑ is a *unifier* of terms t and t' if $t\vartheta \equiv t'\vartheta$ (where $\equiv$ denotes syntactic equality). If two terms are unifiable then they have an idempotent most general unifier which is unique up to renaming. Therefore $\mathrm{mgu}(t_1, t_2)$ will denote such a most general unifier of t_1 and t_2. All the above definitions can be extended to other syntactic expressions in the obvious way.

An atom is an object of the form $p(t_1, ..., t_n)$ where $p \in \Pi$, $t_1, ..., t_n \in T$. A (definite) *clause* is a formula of the form $H: -A_1, ..., A_n$ with $n \geqslant 0$, where H (the *head*) and $A_1, ..., A_n$ (the *body*) are atoms.": $-$" and "," denote logic implication and conjunction respectively, and all variables are universally quantified. If the body is empty the clause is called a *unit clause* and a *goal* is a conjunction of atoms $A_1, ..., A_m$. A *program* is a set of (definite) clauses. $\tilde{t}$, $\tilde{X}$ denote tuples of terms and of *distinct* variables respectively, while $\tilde{B}$ denotes a (possibly empty) conjunction of atoms $B_1, ..., B_n$.

The ordinal powers f^α of a monotonic function f on a complete lattice are defined as usual, namely $f^0(x) = x$, $f^{\alpha+1}(x) = f(f^\alpha(x))$ for any ordinal α and $f^\gamma(x) = \bigsqcup_{\alpha<\gamma} f^\alpha(x)$ for γ limit ordinal. We also use the standard notation $f \uparrow \alpha = f^\alpha(\bot)$, where $\bot$ is the bottom element of the lattice. For f, g generic functions defined on D, for any $x \in D$ we denote $f(x) \cup g(x)$ by $(f+g)(x)$ and $f = g$ means extensional equality, i.e. for all $x \in D$, $f(x) = g(x)$. A function f_P obtained by a program P (such as the immediate consequence operator) induces an equivalence on programs in the obvious way, namely P and Q are equivalent wrt f_P iff $f_P = f_Q$.

If $\tilde{B}$ is the conjunction $B_1, ..., B_n$, we use both the notations $\{|B_1, ..., B_n|\}$ and $\{|\tilde{B}|\}$ to denote the multiset containing the elements $B_1, B_2, ...$ and B_n. By $\mathrm{Var}(E)$ and $\mathrm{Pred}(E)$ we denote respectively the set of variables and the

set of predicates occurring in the expression E. Ground(E) denotes the set of ground instances of E.

An *Herbrand interpretation* I for a program P is a set of ground atoms. The intersection of all the Herbrand models of a program P is a model (the least Herbrand model) which is usually considered *the declarative* semantics of P. Such a least model in the following will be denoted by $M(P)$.

Definite clauses have a natural computational reading based on the resolution procedure. The specific resolution strategy called SLD can be described as follows. Let $G = A_1, ..., A_k$ be a goal and $C = H : -\tilde{B}$ be a (definite) clause. G' is *derived* from G and C by using ϑ iff there exists an atom A_m, $1 \leqslant m \leqslant k$ such that $\vartheta = \text{mgu}(A_m, H)$ and $G' = (A_1, ..., A_{m-1}, \tilde{B}, A_{m+1}, ..., A_k)\vartheta$. Given a goal G and a program P, an *SLD-derivation* (or simply a derivation) of $P \cup \{G\}$ (of G in P) consists in a (possibly infinite) sequence of goals $G_0, G_1, G_2, ...$ called resolvents, together with a sequence $C_1, C_2, ...$ of variants of clauses in P which are *renamed apart* (i.e., such that C_i does not share any variable with $G_0, C_1, ..., C_{i-1}$) and a sequence $\vartheta_1, \vartheta_2, ...$ of idempotent mgu's such that $G_0 = G$ and, for $i \geqslant 1$, each G_i is derived from G_{i-1} and C_i by using ϑ_i. An *SLD-refutation* of $P \cup G$ is a finite SLD-derivation of $P \cup G$ which has the empty clause $\square$ as the last goal in the derivation.

Following [1] we define a *selection rule* R as a function which when applied to a "history" containing all the clauses and the mgu's used in the derivation $G_0, G_1, ..., G_i$, returns an atom in G_i. Such an atom is the selected atom in G_i. Given a selection rule R, an SLD-derivation is called via R if all the selections of atoms in the resolvents are performed according to R. Finally an SLD-derivation is fair if it is either finite or every atom appearing in it is eventually selected. A rule R is *fair* if any SLD-derivation via R is fair.

In the following $G \overset{\vartheta}{\rightarrow}_{P,R} \tilde{B}$ denotes a finite SLD-derivation of $P \cup \{G\}$ via selection rule R, which has length $\geqslant 1$, where ϑ is the composition of the mgu's introduced and $\tilde{B}$ is the last resolvent in the derivation. If R is omitted, we mean that any selection rule can be used (and the definition is independent from the selection rule). The *computed answer substitution* of a refutation $G \overset{\vartheta}{\rightarrow}_P \square$ is the substitution obtained by the restriction of ϑ to the variables occurring in G. $G \overset{\beta}{\mapsto}_P \square$ will denote explicitly the refutation of G in P with computed answer substitution β.

2. OBSERVABLES, COMPOSITION OPERATORS AND EQUIVALENCES

The concrete operational semantics of (logic) programs can be specified by means of a set of inference rules which specify how derivations are made and by defining which are the "observables" we are interested in. In pure logic programming, we can be interested in different observable properties such as successful derivations, finite failures, computed answer substitutions, and partial computed answer substitutions. Therefore a program can have different concrete operational semantics depending on which properties are observed. In this section we introduce the various observables that we will consider in the following and we show the relation among them.

As previously mentioned, a given choice of the observable x induces an "observational" equivalence $\approx_x$ on programs. Namely $P \approx_x Q$ iff P and Q are observationally indistinguishable according to x. When composition of programs also is taken into account, for a given observable property we can obtain different equivalences depending on which kind of program composition we consider. Given an observable x and a syntactic program composition operator $\circ$, the induced equivalence $\approx_{(\circ, x)}$ is defined as follows. $P \approx_{(\circ, x)} Q$ iff for any program R, $P \circ R$ and $Q \circ R$ are observationally indistinguishable according to x (i.e., iff P and Q are observationally indistinguishable under any possible context allowed by the composition operator). The equivalence $\approx_x$ can be considered as a specific case of $\approx_{(\circ, x)}$ (obtained by considering, for any R, $P \circ R = P$).

A semantics $\mathscr{S}(P)$ for a logic program P is correct wrt $\approx_{(\circ, x)}$, if $\mathscr{S}(P) = \mathscr{S}(Q)$ implies $P \approx_{(\circ, x)} Q$. $\mathscr{S}(P)$ is fully abstract wrt $\approx_{(\circ, x)}$ when the converse of the previous implication holds also. A semantic $\mathscr{S}(P)$ is compositional wrt the program composition operator $\circ$, if the semantics of the composition of programs P and Q can be obtained from the semantics of P and the semantics of Q; i.e., if for a suitable function f, $\mathscr{S}(P \circ Q) = f(\mathscr{S}(P), \mathscr{S}(Q))$.

If $\mathscr{S}(P)$ is correct wrt $\approx_x$ and is compositional wrt $\circ$, then $\mathscr{S}(P)$ is also correct wrt $\approx_{(\circ, x)}$. Note that $\mathscr{S}(P)$ can be any semantics (fixpoint, model-theoretic), including a formal operational semantics such as the one defined by the success set [28].

If we are concerned with the input/output behavior of programs we should just observe computed answers. However, there are tasks, such as program analysis and optimization, where we are forced to observe and take into account other features of the derivation. In principle, one could be interested in the complete information about the SLD-derivation, namely the sequences of goals, most general unifiers, and variants of clauses. The *resultants*, introduced in [29] in the framework of partial evaluation, are a compact representation of the relation between the initial goal G and the current "$\langle$*resolvent, substitution*$\rangle$ pair in a SLD derivation of G, where the *substitution* is the (restriction to Var(G) of the) composition of the mgu's computed in the SLD derivation from G to the *resolvent*. The resultant for the derivation $G \overset{\vartheta}{\rightarrow}_{P,R} \tilde{B}$ is then the formula $G\vartheta : -\tilde{B}$ and we say that ϑ is the substitution associated to the resolvent $\tilde{B}$.

Resultants are useful (see [1]) to formalize the properties of SLD-resolution. As we will show in the next section, our semantic scheme models the set of the resultants for any

selection rule. We will then derive as instances of the scheme other semantics which model (in some cases compositionally) more abstract observables. These observables are:

- *partial answers* (denoted by pa), which are the substitutions associated to a resolvent in any SLD-derivation, and *correct partial answers* (denoted by cpa), which are the substitutions associated to a resolvent in any SLD-refutation. The knowledge about partial answers is important for program analysis [11], to characterize the semantics of concurrent languages [16], and to characterize universal termination, which in turn is useful for the semantics of PROLOG [2, 4].
- *call patterns* (denoted by pt), which are the atoms (procedure calls) selected in any SLD-derivation, and *correct call patterns* (denoted by cpt), which are the atoms (procedure calls) selected in any SLD-refutation. Call patterns make it possible to derive properties of procedure calls, which are clearly relevant to program optimization and play an important role in most program analysis frameworks based on abstract interpretation (see [13] for a recent broad overview).
- *computed answers* (denoted by ca), which are the substitutions associated to the last node in an SLD-refutation, and
- *successful derivations* (denoted by s), where we just observe successful termination.

In the following sections we will obtain, as instances of the general scheme, a semantics (in some cases compositional) for each one of the previous observables. Figure 1 shows how the various semantics are mutually related in terms of abstraction ($\mathscr{F}_1 \to \mathscr{F}_2$ means that $\mathscr{F}_2$ is an abstraction of $\mathscr{F}_1$, while semantics contained in the same box do coincide).

- $\mathscr{F}$ is the scheme (resultants semantics),
- $\mathscr{F}_{\mathrm{pt}}$ ($\mathscr{F}_{\mathrm{cpt}}$) is the (correct) call patterns semantics,
- $\mathscr{F}_{\mathrm{pa}}$ ($\mathscr{F}_{\mathrm{cpa}}$) is the (correct) partial answers semantics,
- $\mathscr{F}_{(\Pi,\mathrm{ca})}$ is the compositional computed answers semantics,
- $\mathscr{F}_{\mathrm{ca}}$ is the (non-compositional) computed answers semantics, i.e., the s-semantics [17],
- $\mathscr{F}_{(\Pi,s)}$ is the compositional successful derivations semantics,
- $\mathscr{F}_s$ is the (non-compositional) successful derivations semantics, i.e. the least Herbrand model M.

As explained in the following section, each semantics $\mathscr{F}_i$ is obtained by setting a parameter in the scheme, according to the corresponding observational equivalence $\approx_i$. Moreover each $\mathscr{F}_i$ is correct wrt the corresponding $\approx_i$. In several cases full abstraction also is obtained. A detailed picture summarizing the various results will be shown in Section 7.

2.1. *Observational Equivalences and Their Relations*

We formally define now the observational equivalences that we will consider and we show their relative strength.

Computed answers and successful derivations are known to be independent of the selection rule. This property is based on the switching lemma [1] and on the fact that these observables are obtained from successful derivations, where all the atoms have been evaluated. This is not the case for partial answers which therefore depend on the selection rule. The following is a simple example.

EXAMPLE 2.1. The goal $r(X, Y)$ in the program $P = \{p(a).\ q(b).\ r(X, Y)\colon -p(X), q(Y).\}$ has (correct) partial answers $\vartheta = \{X/a\}$ and $\vartheta' = \{X/a, Y/b\}$ by using the leftmost selection rule. It has partial answers $\gamma = \{Y/b\}$ and ϑ' by using the rightmost selection rule.

An analogous example shows that call patterns depend on the selection rule also. A semantics modeling these observables, for a given selection rule, is obtained in [22] as a natural extension of the constructions discussed in this paper. For the sake of simplicity, in the following we consider only notions which are independent of the selection rule. Therefore, in the case of partial answers and call patterns, we introduce the independence in the definition as follows.

DEFINITION 2.2. Let P be a program and G be a goal. Consider the condition

(i) there exists a selection rule R and a derivation $G \overset{\vartheta}{\to}_{P,R} \tilde{B}$.

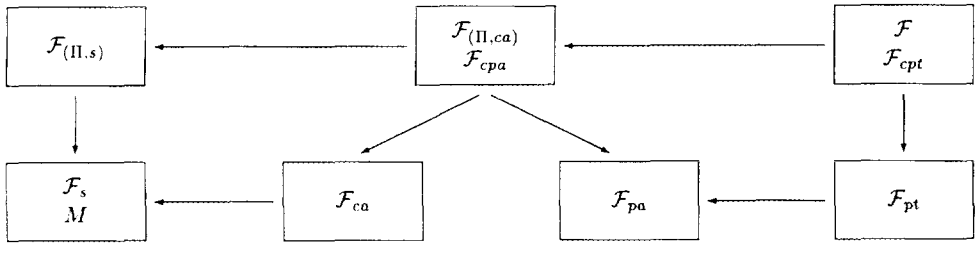

Fig. 1. Relations among various instances of the semantic scheme.

Then we define

1. ϑ is a partial answer (for G in P) iff either $\vartheta = \varepsilon$ or (i) holds and $\vartheta = \gamma_{|\mathrm{Var}(G)}$.

2. ϑ is a correct partial answer iff (i) holds, $\tilde{B}$ has a refutation in P, and either $\vartheta = \varepsilon$ or $\vartheta = \gamma_{|\mathrm{Var}(G)}$.

3. A is a call pattern for G in P iff either A is the atom selected by R in G or (i) holds and A is selected by R in $\tilde{B}$.

4. A is a correct call pattern for G in P iff (i) holds, $\tilde{B}$ has a refutation in P, and A is the atom selected by R either in G or in $\tilde{B}$.

Note that computed answers are a special case of (correct) partial answers. Furthermore, given a correct partial answer ϑ, there exists a computed answer, and therefore (by soundness of SLD resolution) a correct answer γ such that $\vartheta \leqslant \gamma$. In other terms, if ϑ is a correct partial answer for the goal G in program P, then $\exists X.G\vartheta$, where $X = \mathrm{Var}(G\vartheta)$, is a logical consequence of P. This is not the case for generic partial answers.

The only notion of program composition we will consider in the following is a generalization of program union defined as follows.

DEFINITION 2.3 [8]. An Ω-open program is a logic program P together with a set $\Omega \subseteq \Pi$ of predicate symbols. If P and Q are Ω-open programs and

$$(\mathrm{Pred}(P) \cap \mathrm{Pred}(Q)) \subseteq \Omega,$$

then $P \cup_\Omega Q$ is defined as the Ω-open program $P \cup Q$. Otherwise $P \cup_\Omega Q$ is not defined.

An Ω-open program P can be composed, by means of $\cup_\Omega$, with other programs which may further specify the predicates in Ω. The set of predicates Ω specifies which predicates in P can be shared by the other programs. Note that if $\Omega = \Pi$ then we obtain the standard union, while if $\Omega = \varnothing$ the composition is allowed only on programs which do not share predicate symbols. The previous definition is a simplified version of that one in [8], where different sets of predicates are used for P and Q. Note that $\cup_\Omega$ as previously defined is associative, while this is not the case for the original definition in [8].

The combination of the above defined six observables with the $\cup_\Omega$ composition operator gives six observational equivalences. We list below their definitions.

DEFINITION 2.4. Let P and Q be Ω-open programs, G be a goal, and W denote a program such that $P' = P \cup_\Omega W$ and $Q' = Q \cup_\Omega W$ are defined. Assume that $x \in \{\mathrm{s}, \mathrm{ca}, \mathrm{pa}, \mathrm{cpa}, \mathrm{pt}, \mathrm{cpt}\}$. Then we define

$P \approx_{(\Omega, x)} Q$ iff i_x holds for any G and for any W,

where the conditions i_x are defined as follows:

i_s: G has a refutation in P' iff G has a refutation in Q',

i_{ca}: G has the same set of computed answers in P' and in Q',

i_{pa} (i_{cpa}): G has the same set of partial answers (correct partial answers) in P' and in Q',

i_{pt} (i_{cpt}): G has the same set of call patterns (correct call patterns) in P' and in Q'.

As shown by the following proposition, the case $\Omega = \varnothing$ is equivalent to considering no composition at all.

PROPOSITION 2.5. *Let P_1, P_2 be Ω-open programs and let $x \in \{\mathrm{s}, \mathrm{ca}, \mathrm{pa}, \mathrm{cpa}, \mathrm{pt}, \mathrm{cpt}\}$. Then $P_1 \approx_{(\varnothing, x)} P_2$ iff $P_1 \approx_x P_2$.*

Proof. For the non trivial implication, assume that $P_i \cup_\varnothing Q$ is defined for $i = 1, 2$. Then P_i and Q do not share any predicate symbol. Since all the definitions of the observables are independent from the selection rule, we can assume that any derivation for a goal G is performed by first evaluating the atoms which share predicate symbols with P_i and then the remaining atoms. The thesis then is a straightforward consequence of the hypothesis $P_1 \approx_x P_2$ and of the definitions of x. ∎

In order to simplify the notation, in the following we will denote $\approx_{(\varnothing, x)}$ by $\approx_x$. Moreover, we will consider only the $\approx_x$ versions of the equivalences in the case of partial answers and call patterns. The more general case can be treated in a similar way.

The various equivalences are related by a partial ordering, shown in Fig. 2, which indicates their relative strength. The related formal result is given by Proposition 2.8, where $\overset{*}{\to}$ denotes the transitive closure of the relation $\to$.

Proposition 2.8 uses the following result whose proof could be given directly by using previous definitions. However, for the sake of simplicity, we will give a proof in the Appendix which uses some results proved in later sections.

PROPOSITION 2.6. *Let P and Q be programs.*

(i) *If $P \approx_{\mathrm{cpa}} Q$ then $P \approx_s Q$.*

(ii) *If $P \approx_{\mathrm{cpt}} Q$ then $P \approx_s Q$.*

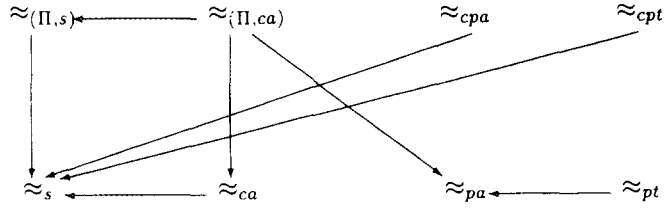

Fig. 2. Relations among observational equivalences.

(iii) *If* $P \approx_{(H, ca)} Q$ *then* $P \approx_{pa} Q$.

(iv) *If* P *and* Q *are finite programs defined on a signature which contains infinitely many unary function symbols,* $P \approx_{pt} Q$ *implies* $P \approx_{pa} Q$.

Note that, as shown by the following example, the hypothesis on the signature is essential for the fourth case of Proposition 2.6.

EXAMPLE 2.7. Let P and Q be programs defined on a signature which contains only the predicate symbol p and no function symbol of arity greater or equal to 1.

$$P = \{ p(X, a): -p(a, a). \quad p(a, X): -p(a, a). \}$$

$$Q = \{ p(X, a): -p(a, a). \quad p(a, X): -p(a, a). \quad p(a, a): -p(a, a) \}$$

Then $P \approx_{pt} Q$ and $P \not\approx_{pa} Q$ (consider the goal $p(X, Y)$).

We can give now the result summarizing the various relations among the equivalences.

PROPOSITION 2.8. *Let* $\approx_i$ *and* $\approx_j$ *be two equivalences and* $\rightarrow$ *be the relation shown in Fig. 2. If* $\approx_i \overset{*}{\rightarrow} \approx_j$ *then the (strict) inclusion* $\approx_i \subset \approx_j$ *holds. Moreover no arrow is omitted (apart from those which can be obtained by transitivity), i.e.,* $\approx_i \subseteq \approx_j$ *implies* $\approx_i \overset{*}{\rightarrow} \approx_j$.

Proof. The first part of the proof is straightforward by definitions, Proposition 2.6, and Example 2.9 which shows that the inclusions are strict. Counter examples which prove the (contrapositive of the) second part are given in Example 2.9. ∎

EXAMPLE 2.9. We assume that all the following programs are defined on a signature containing the constant symbols a and b and the function symbol f.

$$P_1 = \{ p(X). \}$$

$$P_2 = \{ p(X): -p(X). \}$$

$$P_3 = \{ p(X). \quad p(a). \}$$

$$P_4 = \{ p(X). \quad q(X): -r(X). \}$$

$$P_5 = \{ p(X). \quad p(f(X)): -q(Y), q(a). \quad p(f(a)). \quad q(a). \}$$

$$P_6 = \{ p(X): -q(Y), q(a). \quad p(f(X)): -q(X), q(Y), q(a). \quad p(f(a)). \quad q(a). \}$$

Let us consider the goals $p(X)$ and $q(X)$. It is easy to verify that

$P_1 \approx_{(H, s)} P_3$	and	$P_1 \not\approx_{ca} P_3$	(i.e., $\approx_{(H, s)} \not\subseteq \approx_{ca}$)
$P_1 \approx_x P_2$	and	$P_1 \not\approx_s P_2$,	for $x \in \{pa, pt\}$
$P_5 \approx_x P_6$	and	$P_5 \not\approx_{ca} P_6$,	for $x \in \{cpa, cpt\}$
$P_1 \approx_x P_4$	and	$P_1 \not\approx_{(H, s)} P_4$,	for $x \in \{ca, cpa, cpt\}$

(take the union with $Q = \{ r(a) \}$).

Moreover, the programs

$$Q_1 = \{ r(a, b). \}$$

$$Q_2 = \{ r(a, X): -r(a, X), r(a, Y). \quad r(a, b). \}$$

$$Q_3 = \{ r(c, X): -r(c, X). \quad r(a, b). \}$$

$$Q_4 = \{ r(a, X): -r(a, X), r(a, Y). \quad r(a, b): -r(a, X). \}$$

$$Q_5 = \{ p(a): -q(a). \quad q(a). \}$$

$$Q_6 = \{ p(a). \quad q(a). \}$$

$$Q_7 = \{ p(f(X)): -p(c). \quad p(f(X)): -p(f(X)). \quad p(f(a)). \}$$

$$Q_8 = \{ p(f(X)): -p(c). \quad p(f(a)). \}$$

$$Q_9 = \{ p(X): -q(Y), q(a). \quad q(a). \}$$

$$Q_{10} = \{ p(X): -q(X), q(Y), q(a). \quad q(a). \}$$

show that

$Q_1 \approx_{ca} Q_2$	and	$Q_1 \not\approx_x Q_2$,	for $x \in \{cpa, cpt, pa, pt\}$ (consider the goal $r(X, Y)$)
$P_1 \approx_{(H, s)} P_3$	and	$P_1 \not\approx_x P_3$,	for $x \in \{cpa, cpt, pa, pt\}$ (consider the goal $p(X), p(X)$)
$Q_2 \approx_x Q_4$	and	$Q_2 \not\approx_y Q_4$,	for $x \in \{pa, pt\}$, $y \in \{cpa, cpt\}$ (consider the goal $r(X, Y)$)
$Q_1 \approx_x Q_3$	and	$Q_1 \not\approx_y Q_3$,	for $x \in \{cpa, cpt\}$, $y \in \{pa, pt\}$ (as before)
$Q_5 \approx_{(H, ca)} Q_6$	and	$Q_5 \not\approx_x Q_6$,	for $x \in \{pt, cpt\}$ (consider the goal $p(X)$)
$Q_7 \approx_{(H, ca)} Q_8$	and	$Q_7 \not\approx_{cpa} Q_8$,	(as before)
$Q_9 \approx_{cpt} Q_{10}$	and	$Q_9 \not\approx_{cpa} Q_{10}$,	(as before)

The above examples and the transitivity of the $\overset{*}{\rightarrow}$ relation show that if we cannot obtain $\approx_x \overset{*}{\rightarrow} \approx_y$ from the relation $\rightarrow$ in Fig. 2, then $\approx_x \not\subseteq \approx_y$ (i.e., $\approx_x$ is not finer than $\approx_y$).

3. A SEMANTIC SCHEME

The scheme we propose in this section is a generalization of the open semantics introduced in [9, 8] to obtain compositionality wrt program union. The standard semantics based on atoms are not compositional wrt union of programs. Consider for instance the programs $P = \{q(a).\ p(X)\colon -r(X).\}$, $Q = \{q(a).\}$ and $R = \{r(a).\}$. The least Herbrand model semantics $M(P)$ identifies P and Q, since $M(P) = M(Q) = \{q(a)\}$. However, $M(P \cup R) \neq M(Q \cup R)$. In order to obtain the semantics of the union $P \cup R$ from those of the components, the semantics of P should contain also the information given by the clause $p(X)\colon -r(X)$. The open semantics was then defined on domains containing equivalence classes of sets of (equivalence classes of) clauses (called π-interpretations).

If we abstract from the specific equivalences in [9, 8], the open semantics can be viewed as a semantic framework for correctly modeling $\approx_{(\cdot,\cdot,x)}$ equivalences. We give below the general definitions parametrically wrt an equivalence relation $\sim$ on sets of clauses used to build the semantic domain. Depending on the composition operator $\circ$ and the observable x, the semantics for $\approx_{(\cdot,\cdot,x)}$ can be obtained as an instance of the general scheme by defining a specific equivalence relation $\sim_{(\cdot,\cdot,x)}$ on sets of clauses and therefore a specific semantic domain.

We denote by $\equiv_c$ the equivalence on clauses defined as follows:

$$H\colon -A_1, \ldots, A_n \equiv_c K\colon -B_1, \ldots, B_n$$

iff there exists a renaming ρ such that $H\rho = K$ and the multisets $\{|A_1\rho, \ldots, A_n\rho|\}$ and $\{|B_1, \ldots, B_n|\}$ are equal. Note that on unit clauses $\equiv_c$ is just variance.

DEFINITION 3.1. We denote by $\mathscr{C}$ the set of the $\equiv_c$ equivalence classes of clauses (on the given signature). Moreover, given a set of predicates Ω we define

$$\mathscr{C}^{\Omega} = \{H\colon -p_1(\tilde{t}_1), \ldots, p_n(\tilde{t}_n) \in \mathscr{C} \mid \text{ for } i = 1, \ldots, n, p_i \in \Omega\}$$

$$\mathscr{A} = \{p(\tilde{t}) \mid p(\tilde{t}) \in \mathscr{C}\}$$

$$\mathscr{A}^{\Omega} = \{p(\tilde{t}) \mid p(\tilde{t}) \in \mathscr{C} \text{ and } p \in \Omega\}$$

Notation. To simplify the notation we will denote both a clause c and its $\equiv_c$ equivalence class by c. We assume every syntactic element of an $\equiv_c$ equivalence class to be implicitly renamed apart, i.e., that it does not share any variable with any other expression in the given context. Moreover, each syntactic operator f on clauses will be considered also as an operator on $\wp(\mathscr{C})$, still denoted by f, obtained by applying f to the (renamed apart) elements of the $\equiv_c$ equivalences classes. This is correct since all the syntactic operators (and derivations) use clauses which are renamed apart.

Let us now introduce the basic syntactic operator $\mathrm{unf}_Q(P)$ which denotes the result of the *unfolding* of P wrt Q, i.e. the result of the parallel rewriting of the bodies of clauses in P by using clauses in Q.

Using $\mathrm{unf}_Q(P)$ we define also the operator Γ_P which will later be used to construct the general immediate consequence operator $\mathscr{T}_P$. Given a set of clauses Q, $\Gamma_P(Q)$ generates all the clauses obtained by "partially" unfolding P wrt Q, i.e., it generates also those clauses obtained by rewriting a (possibly empty) subset of the atoms in the bodies. Note that the arguments in $\Gamma_P(Q)$ are exchanged wrt $\mathrm{unf}_Q(P)$ since, analogously to the case of the T_P operator, in $\Gamma_P(Q)$ the set of clauses Q is the subject of the operation while P is a qualification.

In the following we use the notation $\mathrm{Id}_{\Pi} = \{p(\tilde{X})\colon -p(\tilde{X}) \mid p \in \Pi\}$.

DEFINITION 3.2. Let P, Q be sets of clauses. Then we define

$$\begin{aligned}\mathrm{unf}_Q(P) = \{(A\colon -\tilde{B}_1, \ldots, \tilde{B}_n)\vartheta \mid\ & \exists A\colon -B_1, \ldots, B_n \in P, \quad \text{for } i = 1, \ldots, n, n \geq 0,\\ & \exists H_i\colon -\tilde{B}_i \in Q \quad \text{renamed apart}\\ & \text{such that } \vartheta = \mathrm{mgu}((B_1, \ldots, B_n), (H_1, \ldots, H_n))\}.\end{aligned}$$

Moreover, we define $\Gamma_P(Q) = \mathrm{unf}_{Q \cup \mathrm{Id}_{\Pi}}(P)$.

Let us now give the formal definition of π-interpretation. We denote by $[I]$ an equivalence class containing I.

DEFINITION 3.3. Let $\sim$ be an equivalence relation on $\wp(\mathscr{C})$ which verifies the following properties, where N denotes a set of natural numbers

(i) if, for all $n \in N$, $I_n, J_n \subseteq \mathscr{C}$ and $I_n \sim J_n$, then $\bigcup_{n \in N} I_n \sim \bigcup_{n \in N} J_n$,

(ii) if $I \sim J$ then, for any program P, $\Gamma_P(I) \sim \Gamma_P(J)$.

A π-interpretation is a $\sim$-equivalence class $[I]$ where $I \subseteq \mathscr{C}$. We denote by $\Im$ the set of all the π-interpretations and we define $\iota(I) = a$ where a is any element in $I \in \Im$.

All the definitions which use elements from $\Im$ are parametric wrt an equivalence $\sim$ which satisfies (i) and (ii). However, in the remainder of this section, we omit the $\sim$ index in order to simplify the notation. Strictly speaking, if $\iota(I) = a$ then a is a set of $\equiv_c$ equivalence classes of clauses. According to the previous note, we will consider $\iota(I)$ also as a *set of syntactic clauses* which have been *renamed apart*.

In Definition 3.3 we require $\sim$ to be a congruence wrt infinite unions and wrt the Γ operator. The first restriction

is needed to obtain the complete lattice structure that we describe below for the set $\mathfrak{I}$. The second one will guarantee the correctness of the definition of our general fixpoint semantics. All the definitions of $\sim$ we will introduce in the following sections satisfy these constraints.

DEFINITION 3.4. Let $X \subseteq \mathfrak{I}$. We define $\sqcup X = [\bigcup_{I \in X} \iota(I)]$ and for $I, J \in \mathfrak{I}$, $I \sqsubseteq J$ iff $I \sqcup J = J$.

For $X \subseteq \mathfrak{I}$, $\sqcup X$ is well defined since $\sim$ is a congruence wrt (infinite) union. Hence the definition of $\sqcup X$ does not depend on the elements $\iota(I)$ chosen for $I \in X$.

LEMMA 3.5. *The relation $\sqsubseteq$ is an ordering on $\mathfrak{I}$. Moreover, $(\mathfrak{I}, \sqsubseteq)$ is a complete lattice (with $\sqcup$ as lub and $[\varnothing]$ as bottom).*

We can now define the general semantics scheme in terms of π-interpretations and hence parametrically wrt $\sim$. We give two equivalent characterizations. The top-down one has a definition in the style of an operational semantics, while the bottom-up one is based on the fixpoint of a general immediate consequences operator. Let us first define the top-down semantics $\mathscr{C}(P)$.

DEFINITION 3.6 (Operational Semantic Scheme). Let P be a program, R be a fair selection rule and assume $P^* = P \cup \mathrm{Id}_H$. We define $\mathscr{C}(P) \in \mathfrak{I}$ as

$$\mathscr{C}(P) = [\{ p(\tilde{X})\vartheta\gamma\colon -\tilde{B} \in \mathscr{C} \mid \text{there exists a derivation } p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{P,R} \tilde{D} \overset{\gamma}{\rightsquigarrow}_{P^*,R} \tilde{B} \}].$$

Note that $\mathscr{C}(P)$ is a π-interpretation and it is the (equivalence class of the) set of all the *resultants* [29], obtained from goals of the form $p(\tilde{X})$ in P for any possible selection rule. Indeed the previous definition is independent from the *fair* selection rule R and, because of the "trick" of adding Id_H, it considers derivations for any selection rule. This is formally stated by the following.

PROPOSITION 3.7 [8]. *Let R be a fair selection rule and let $P^* = P \cup \mathrm{Id}_H$. Then there exists a rule R' and there exists a derivation $p(\tilde{X}) \overset{\sigma}{\rightsquigarrow}_{P,R'} \tilde{B}$ iff there exists a derivation $p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{P,R} \tilde{D} \overset{\gamma}{\rightsquigarrow}_{P^*,R} \tilde{B}$ such that $p(\tilde{X})\sigma = p(\tilde{X})\vartheta\gamma$.*

To obtain a fixpoint definition we first define the immediate consequences operator $\mathscr{T}_P$ on the lattice of π-interpretations. $\mathscr{T}_P$ is defined simply as the semantic counterpart of the syntactic operator Γ_P. In the following id denotes the identity function.

DEFINITION 3.8. Let P be a program. Then we define $\mathscr{T}_P\colon \mathfrak{I} \to \mathfrak{I}$ as

$$\mathscr{T}_P(I) = [\Gamma_P(\iota(I))].$$

Moreover, we denote $(\mathscr{T}_P + \mathrm{id})^\omega$ by $[\![P]\!]$.

$\mathscr{T}_P(I)$ is well defined, i.e. its definition is independent from the element chosen in the equivalence class I, because $\sim$ is a congruence wrt Γ. Our definition of $[\![P]\!]$ is the generalization of the (ground) closure operator $[\![P]\!]^s = (T_P + \mathrm{id})^\omega(X)$ (where T_P is the standard immediate consequences operator [36]) introduced in [26] to denote the function corresponding to deductions in any number of step. The following lemma shows the result needed to define as usual the least fixpoint semantics.

LEMMA 3.9. *Let P be a program. Then $\mathscr{T}_P$ is continuous on $(\mathfrak{I}, \sqsubseteq)$ and $\mathscr{T}_P \uparrow \omega$ is the least fixpoint of $\mathscr{T}_P$.*

DEFINITION 3.10 (Fixpoint Semantic Scheme). Let P be a program. We define $\mathscr{F}(P) = \mathscr{T}_P \uparrow \omega$.

Because of the previously mentioned ability of $\Gamma_P(I)$ (and therefore of $\mathscr{T}_P$) to produce also the result of partial unfoldings, $\mathscr{F}(P)$ gives a bottom-up description of partial derivations, i.e., it contains also the intermediate results of non-terminated (and possibly non-terminating) computations. Indeed, as shown by the following theorem, no matter which specific $\sim$ equivalence is used, we can prove the equality of the top-down and the bottom-up constructions. This general result simplifies the treatment in specific cases since it is usually easier proving the congruence requirements on $\sim$ rather than proving the stated equality.

THEOREM 3.11 (Equivalence). *Let P be a program and let $\sim$ be an equivalence on $\wp(\mathscr{C})$ which satisfies conditions 1 and 2 of Definition 3.3. Then $\mathscr{F}(P) = \mathscr{C}(P)$.*

Proof. When $\sim$ is identity on $\wp(\mathscr{C})$ the theorem has been proved in [8] (Corollary 4.17). Obviously, for any set $\mathscr{A}$ and any $A, B \in \wp(\mathscr{A})$ if $A = B$ then the $\sim$-equivalence classes $[A]$ and $[B]$ are equal for any equivalence $\sim$ on $\mathscr{A}$. Therefore the thesis holds. ∎

By instantiating $\sim$ to a specific equivalence $\sim_{(\circ, x)}$, which depends on the composition operator $(\circ)$ and the observable (x), we can obtain suitable $\mathscr{T}_P$ operators and (equivalent operational and fixpoint) semantics for the corresponding $\approx_{(\circ, x)}$ equivalences. The composition operators which can be treated in this way are essentially variants of union (like $\cup_\Omega$ and the operators considered in [8]).

When considering as $\sim$ the identity on $\wp(\mathscr{C})$, as shown by Proposition 3.12, we obtain a kind of "collecting semantics" which correctly models resultants. This gives the maximum amount of information on computations since it allows us to observe all the internal details of SLD derivations.

In the following, given a goal $G = A_1, \ldots, A_k$ we say that G' is a subgoal of G if $G = A_{j_1}, \ldots, A_{j_n}$, $n \geq 0$ where $\{j_1, \ldots, j_n\} \subseteq \{1, \ldots, k\}$ and $j_k \neq j_l$ for $k \neq l$. Moreover, if λ is an empty goal then $\mathrm{mgu}(\lambda, \lambda)$ denotes the empty substitution.

PROPOSITION 3.12. *Let P be a program and G be a goal. Assume that the semantic domain is defined by letting $\sim$ be the identity. Then there exist R, ϑ and a derivation $G \overset{\vartheta}{\rightsquigarrow}_{P,R} \tilde{B}$ with $\vartheta_{|\mathrm{Var}(G)} = \gamma$ iff there exist a non-empty subgoal $G' = A_1, \ldots, A_n$ of G and a sequence $\tilde{H}$ of heads of n clauses $H_j: -\tilde{D}_j$ in $i(\mathscr{F}(P))$ such that*

1. $\xi = \mathrm{mgu}(G', \tilde{H})$ *and* $\gamma = \xi_{|\mathrm{Var}(G)}$,
2. $\{|G''\xi|\} = \{|\tilde{B}|\}$ *where G'' denotes the goal obtained from G by replacing each A_j by $\tilde{D}_j$.*

The semantics modeling resultants is clearly correct wrt the equivalence induced by any notion of observability we can consider. However, we are interested in defining, for specific observables, coarser $\sim$ equivalences in order to obtain a more (possibly fully) abstract semantics, while preserving the correctness.

In the following we will then introduce a suitable $\sim_i$ equivalence to obtain a correct (in some cases fully abstract) semantics for any $\approx_i$ equivalence considered in the previous section. For example, weak subsumption equivalence [30] can be used for $\approx_{(\Pi, s)}$, i.e., for program union as composition operator ($\circ = \cup_\Pi$) and successful derivations as observable ($x = s$), but it is too coarse and hence not correct when considering computed answers (see Example 4.14). As previously argued, the semantics arising in abstract interpretation can be obtained as well by considering suitable observables and suitable $\sim_i$ equivalences.

Notation. In the remainder of the paper the instances of the generic constructions $\mathfrak{I}$, $\mathscr{T}_P$, $\mathscr{C}$ and $\mathscr{F}$ which are obtained by using a specific $\sim_i$ equivalence, will be denoted by $\mathfrak{I}_i$, T_P^i, $\mathscr{C}_i$ and $\mathscr{F}_i$, respectively. When the subscripts are omitted we mean that $\sim$ is the identity on $\wp(\mathscr{C})$.

4. THE CASE OF PREDICATE DISJOINT PROGRAMS

In this section we consider the composition of programs which do not share predicates (i.e. $\Omega = \varnothing$). As discussed in Section 2 this is the same as the case of no composition at all. Here the observables we are concerned with are computed answer substitutions and successful derivations. The induced equivalences on programs have been previously denoted by $\approx_{\mathrm{ca}}$ and $\approx_s$. We first show that suitable definitions of $\sim_{\mathrm{ca}}$ and $\sim_s$ allow us to obtain the s-semantics [17] and the least Herbrand model as instances of the scheme. Then we consider the relation of these semantics to $\approx_{\mathrm{ca}}$ and $\approx_s$. Finally we discuss some equivalences based on the T_P^{ca} operator.

Since here we are not concerned with compositions, it is sufficient to extract from each set of clauses I only the information given by the unit clauses contained in I. Two sets of clauses can then be considered equivalent if they contain the same unit clauses (up to variance). Moreover, in the case of successful derivations, we only need the information given by the ground instances of the clauses. Proposition 4.4 will formally prove this fact. We define then $\sim_{\mathrm{ca}}$ and $\sim_s$ as follows. Recall that $\mathscr{A}$ is the set of atoms modulo variance (Definition 3.1) and that Ground(S) denotes the set of the ground instances of the elements in S.

DEFINITION 4.1. Let $I, J \subseteq \mathscr{C}$. We define $I \sim_{\mathrm{ca}} J$ iff $I \cap \mathscr{A} = J \cap \mathscr{A}$. Moreover we define $I \sim_s J$ iff Ground$(I \cap \mathscr{A}) =$ Ground$(J \cap \mathscr{A})$.

The previous definition satisfies the requirements on the $\sim$ equivalences stated in Section 3, as shown by the following.

PROPOSITION 4.2. *Let $I, J \subseteq \mathscr{C}$. For $x \in \{\mathrm{ca}, s\}$ the following properties hold*

(i) *$\sim_x$ is a congruence wrt infinite unions,*

(ii) *if $I \sim_x J$ then $\Gamma_P(I) \sim_x \Gamma_P(I)$.*

Proof. We first consider the case $x = s$. To prove (i) it is sufficient to observe that, given a set of integers N, if $I_n \sim_s J_n$ holds for all $n \in N$ and $I_n, J_n \subseteq \mathscr{C}$, then

$$\begin{aligned}
&\mathrm{Ground}((\textstyle\bigcup_{n\in N} I_n) \cap \mathscr{A}) \\
&\qquad \text{(by definition of Ground and set theory)} \\
&= \textstyle\bigcup_{n\in N} \mathrm{Ground}(I_n \cap \mathscr{A}) \\
&\qquad \text{(by hypothesis)} \\
&= \textstyle\bigcup_{n\in N} \mathrm{Ground}(J_n \cap \mathscr{A}) \\
&\qquad \text{(by definition of Ground and set theory)} \\
&= \mathrm{Ground}((\textstyle\bigcup_{n\in N} J_n) \cap \mathscr{A}).
\end{aligned}$$

To prove (ii) we have to show that if H is a (ground) atom in Ground$(\Gamma_P(I))$ then $H \in$ Ground$(\Gamma_P(J))$. Note that, by definition of Γ_P,

$$\mathrm{Ground}(\Gamma_P(I)) = \mathrm{Ground}(\Gamma_P(\mathrm{Ground}(I)).$$

Assume that H is an atom in Ground$(\Gamma_P(I))$.

The previous equality implies that $H \in$ Ground $(\Gamma_P(\mathrm{Ground}(I)))$. Then there exist a clause $A: -\tilde{B}$ in P and a sequence $\tilde{B}'$ of atoms in Ground(I) such that $\vartheta = \mathrm{mgu}(\tilde{B}, \tilde{B}')$ and H is a ground instance of $A\vartheta$.

By Definition 4.1 the hypothesis $I \sim_s J$ implies that I and J have the same ground instances of unit clauses (atoms). Therefore, by using the same clause and the same sequence as before, $H \in \mathrm{Ground}(\Gamma_P(\mathrm{Ground}(J)))$ and (ii) holds.

Let us now consider the case $x = \mathrm{ca}$. The proof of (i) is straightforward by definition of $\sim_{\mathrm{ca}}$. To prove (ii) we have to show that if $H \in \Gamma_P(I) \cap \mathscr{A}$ then $H \in \Gamma_P(J) \cap \mathscr{A}$ (where H is a variance equivalence class of atoms). If $H \in \Gamma_P(I) \cap \mathscr{A}$, by definition of Γ_P, there exist a clause $A: -\tilde{B} \in P$ and a sequence $\tilde{B}'$ of renamed apart atoms in (the variance equivalence classes in) $I \cap \mathscr{A}$ such that

$\vartheta = \mathrm{mgu}(\tilde{B}, \tilde{B}')$ and $H = A\vartheta$. By Definition 4.1, the hypothesis $I \sim_{ca} J$ implies that I and J have the same (variance equivalence classes of) unit clauses (atoms). Therefore, analogously to the previous case, $A\vartheta \in \Gamma_P(J) \cap \mathscr{A}$ and this completes the proof. ∎

Let us first consider the instances of the general definitions obtained by using $\sim_{ca}$. For any $I \in \mathfrak{I}_{ca}$, the set of unit clauses (modulo variance) of any element $\iota(I)$ can be considered the canonical representative of the equivalence class $[I]$. T_P^{ca} defined in terms of canonical representatives is essentially the immediate consequence operator defined in [17].

More precisely, let us define for $I \in \mathfrak{I}_{ca}$ the canonical representative as $\iota_{ca}(I) = \{A \in \mathscr{A} \mid A \in \iota(I)\}$. Clearly, for any $I, J \in \wp(\mathscr{C})$, if $I \sim_{ca} J$ then $\iota_{ca}(I) = \iota_{ca}(J)$. Hence $\iota_{ca}(I)$ is well defined. The operator $T_P^{s-sem}: \wp(\mathscr{A}) \to \wp(\mathscr{A})$ was originally defined in [17] as $T_P^{s-sem}(X) = \iota_{ca}(\Gamma_P(X))$. The s-semantics is the least fixpoint $T_P^{s-sem} \uparrow \omega$ of such an operator. Proposition 4.3 states the equivalence of the two different formulations. As an obvious consequence, the s-semantics as originally defined is the canonical representative of $\mathscr{F}_{ca}$.

Analogously, in the case of $\sim_s$, the canonical representative $\iota_s(J)$ of $J \in \mathfrak{I}_s$ can be obtained by taking the ground instances of the unit clauses in $\iota(J)$, i.e., $\iota_s(J) = \mathrm{Ground}(\iota_{ca}(J))$. Also in this case, $\iota_s(J)$ is well defined. T_P^s in terms of canonical representative is essentially the standard immediate consequence operator [36] $T_P: \wp(\mathrm{Ground}(\mathscr{A})) \to \wp(\mathrm{Ground}(\mathscr{A}))$ which (by using our notation) can be defined as $T_P(X) = \iota_s(\Gamma_P(X))$. Also in this case, the equivalence of the two formulations is shown by the following proposition, and the least fixpoint of T_P (the least Herbrand model $M(P)$) is the canonical representative of $\mathscr{F}_s(P)$.

PROPOSITION 4.3. *Let $X \in \mathfrak{I}_{ca}$ and $W \in \mathfrak{I}_s$. Then*

$$T_P^{ca}(X) = Y \quad \text{iff } T_P^{s-sem}(\iota_{ca}(X)) = \iota_{ca}(Y)$$

$$T_P^s(W) = Z \quad \text{iff } T_P(\iota_s(W)) = \iota_s(Z)$$

Proof. The proof is straightforward by the definitions. ∎

Proposition 4.4 shows that $\approx_s$ is the same as the equivalence induced by $M(P)$. The mentioned correspondence between $M(P)$ and $\mathscr{F}_s(P)$ implies that the latter semantics is fully abstract wrt $\approx_s$ (Corollary 4.5). Next proposition follows from standard results and (for the second part) from a result in [30]. We give here a direct proof for the sake of completeness. Note that the hypothesis on the signature and on the finiteness of programs is essential for the second part.

PROPOSITION 4.4. *Let P and Q be programs. Then*

$$P \approx_s Q \quad \text{iff } M(P) = M(Q)$$

(i.e., iff P and Q have the same successful derivations for ground goals). Moreover, if P and Q are finite programs defined on a signature which contains infinitely many constant symbols,

$$M(P) = M(Q) \quad \text{iff } A(P) = A(Q)$$

where $A(P) = \{A \in \mathscr{A} \mid P \models A\}$ (i.e. $A(P)$ is the set of atomic logic consequences).

Proof. The equality between the success set (the set of ground goals which have a refutation) and the least Herbrand model is a well known result (see for example [28]). We prove the non trivial implication in the first equivalence. Let us assume that $P \not\approx_s Q$ and $M(P) = M(Q)$. Then there exists a goal G which has a refutation $G \overset{\vartheta}{\mapsto}_P \square$ and which has no refutation in Q. By soundness of SLD resolution, ϑ is a correct answer for G in P. Therefore in [28] there exists a substitution β such that $\vartheta\beta$ is a correct answer for G and $G\vartheta\beta = G'$ is ground. By completeness of SLD resolution and since G' is ground, there exists a refutation $G' \overset{\varepsilon}{\mapsto}_P \square$. Since G' is an instance of G, the hypothesis on Q and the lifting lemma [28] imply that G' has no refutation in Q. Therefore, by definition of SLD resolution, there exists an atom A in G' such that A has a refutation in P and has no refutation in Q. Because of the equality between the success set and the least Herbrand model this means that $A \in M(P) \backslash M(Q)$. We have a contradiction.

As for the second equivalence, first observe that by using the definitions it is easy to show that $M(P) = \mathrm{Ground}(A(P))$ (see also [18]). This proves the implication $A(P) = A(Q) \Rightarrow M(P) = M(Q)$. For the other implication, by the strong soundness and completeness theorem in [10] $A(P) = \{A \mid A \overset{\varepsilon}{\mapsto}_P \square\}$. Assume now that $M(P) = M(Q)$ and there exists an atom A such that A has a refutation with empty computed answers in P and not in Q. Since the signature is infinite, we can consider a substitution ϑ which instantiates all the variables in A to constants which do not appear in Q. Then the goal $A\vartheta$ has a refutation in P and has no refutation in Q. This implies that $A\vartheta \in M(P) \backslash M(Q)$, which contradicts the previous equality and completes the proof. ∎

COROLLARY 4.5. *Let P and Q be programs defined on a signature Σ which contains infinitely many constant symbols. Then $P \approx_s Q$ iff $\mathscr{F}_s(P) = \mathscr{F}_s(Q)$.*

The strong completeness theorem in [17] shows that the s-semantics is fully abstract wrt $\approx_{ca}$. Because of the mentioned correspondence with $\mathscr{F}_{ca}(P)$ we can state such a theorem as follows. The same result was obtained in [24] using a proof theoretic approach.

THEOREM 4.6 [17]. *Let P and Q be programs. Then $P \approx_{ca} Q$ iff $\mathscr{F}_{ca}(P) = \mathscr{F}_{ca}(Q)$.*

4.1. *Equivalences Based on* T_P^{ca}

In order to simplify the notation, in the following we assume that T_P^{ca} is defined as its canonical version T_P^{s-sem} (on $\wp(\mathscr{A})$ as shown before). Consequently we assume also $[\![P]\!]^{ca} = (T_P^{s-sem} + id)^{\omega}$. Analogously, we assume that T_P^s is defined as the original immediate consequences operator T_P (on $\wp(\text{Ground}(\mathscr{A}))$). All the results that we will give in the following can be stated for the operators obtained as instances of the generic $\mathscr{T}_P$ just by changing the notation. We consider the relation among several equivalences based on T_P^{ca}, thus extending to the "non-ground" case some results given in [30] for T_P^s and $[\![P]\!]^s$.

The following proposition shows that the "nonground" equivalences are in general strictly finer than their ground versions. The relations between the equivalences for the non-ground case are shown in Proposition 4.9.

PROPOSITION 4.7. *Let P be a program. Then*

1. *the equivalence induced by* T_P^{ca} *is strictly finer than the equivalence induced by* T_P^s,

2. *the equivalence wrt* $[\![P]\!]^{ca}$ *is strictly finer than the equivalence wrt* $[\![P]\!]^s$.

Proof. The properties are a straightforward consequence of the definitions, by noting that for any set of ground atoms, $T_P^s(X) = \text{Ground}(T_P^{ca}(X))$. Strictness can be shown by considering programs P_1 and P_3 in Example 2.9. ∎

For the ground case, the discrimination power of the various equivalences is stated by the following proposition. Recall that a clause $H_1: -A_1, ..., A_n$ subsumes a clause $H_2: -B_1, ..., B_m$ iff there exists a substitution ϑ such that $H_1\vartheta = H_2$ and $\{A_1\vartheta, ..., A_n\vartheta\} \subseteq \{B_1, ..., B_m\}$. Two sets of clauses P and Q are subsumption equivalent iff for every $c \in P$ there exists $c' \in Q$ such that c' subsumes c and vice versa. P and Q are weakly subsumption equivalent iff $P \backslash \text{Taut}(P)$ and $Q \backslash \text{Taut}(Q)$ are subsumption equivalent, where $\text{Taut}(P)$ denotes the set of tautologies in P. $\models (\models_{HU}) P \Leftrightarrow Q$ means that P and Q have the same (Herbrand) models.

PROPOSITION 4.8 [30]. *Let P, Q be two (finite) programs defined on a signature Σ which contains infinitely many constant symbols. Then the following statements hold:*

1. $T_P^s = T_Q^s$ *iff P and Q are subsumption equivalent,*

2. $T_P^s + id = T_Q^s + id$ *iff P and Q are weakly subsumption equivalent,*

3. *if* $T_P^s + id = T_Q^s + id$ *then* $[\![P]\!]^s = [\![Q]\!]^s$ *and the converse does not hold,*

4. $\models P \Leftrightarrow Q$ *iff* $\models_{HU} P \Leftrightarrow Q$ *iff* $[\![P]\!]^s = [\![Q]\!]^s$,

5. *if* $\models P \Leftrightarrow Q$ *then* $M(P) = M(Q)$ *and the converse does not hold.*

As for the equivalences induced by T_P^{ca} and by $[\![P]\!]^{ca}$, we can give the following similar result. A different characterization of the equivalence $[\![P]\!]^{ca} = [\![Q]\!]^{ca}$ will be given in Section 5.

PROPOSITION 4.9. *Let P, Q be programs. Then*

1. *if* $T_P^{ca} + id = T_Q^{ca} + id$ *then* $[\![P]\!]^{ca} = [\![Q]\!]^{ca}$ *and the converse does not hold,*

2. *if* $[\![P]\!]^{ca} = [\![Q]\!]^{ca}$ *then* $P \approx_{ca} Q$ *and the converse does not hold,*

3. *the equivalence* $\approx_{ca}$ *is different from (neither coarser nor finer than) the equivalences induced by* T_P^s, $T_P^s + id$ *and* $[\![P]\!]^s$.

Proof. The proof of the *if* in 1 and 2 is a straightforward consequence of the definitions. For the converse see Example 4.10. As for point 3, by Example 4.10 (first part), $\approx_{ca}$ is not finer than logical equivalence (i.e., the $[\![P]\!]^s$ equivalence by point 4 of Proposition 4.8) nor finer than the equivalence induced by T_P^s and $T_P^s + id$. Programs P_1 and P_3 in Example 2.9 show that logical equivalence and the equivalences induced by T_P^s and $T_P^s + id$ are not finer than $\approx_{ca}$. ∎

EXAMPLE 4.10. *Let us consider the programs*

$$Q_1 = \{p. \quad\} \qquad Q_2 = \{p.\}$$
$$q: -r.$$

Then $Q_1 \approx_{ca} Q_2$ *while* Q_1 *and* Q_2 *are not logically equivalent and* $[\![Q_1]\!]^{ca} \neq [\![Q_2]\!]^{ca}$. *Consider now*

$$P = \{p: -q. \qquad Q = \{p: -q.$$
$$q: -b. \qquad\qquad p: -b.$$
$$b. \quad\} \qquad\qquad q: -b.$$
$$b. \quad\}$$

Then $[\![P]\!]^{ca} = [\![Q]\!]^{ca}$ *and* $T_P^{ca} + id \neq T_Q^{ca} + id$.

Note that, by 1 of Proposition 4.8 and by 1 of Proposition 4.7, the equivalence induced by the T_P^{ca} operator is strictly finer than subsumption equivalence. An equivalence based on T_P^{ca} which is the same as subsumption equivalence is shown by Corollary 4.13.

DEFINITION 4.11. Let $X, Y \subseteq \mathscr{A}$. Then we define $X \prec Y$ iff for any $A \in X$ there exists $B \in Y$ such that $B \leqslant A$ (i.e. B is more general than A). If f, g are functions defined on $\wp(\mathscr{A})$, $f \prec g$ iff $\forall X \in \wp(\mathscr{A}), f(X) \prec g(X)$.

PROPOSITION 4.12. *Assume that* $P = \{c_1\}$, $Q = \{c_2\}$. *If the clause* c_1 *subsumes* c_2 *then* $T_Q^{ca} \prec T_P^{ca}$.

Proof. We can assume, without loss of generality that the clauses $c_1: H: -A_1, ..., A_m$ and $c_2: K: -B_1, ..., B_n$ are renamed apart. Given the set of atoms X, if $A \in T_Q^{ca}(X)$

then, by definition of T_Q^{ca}, there exist $C_1, ..., C_n \in X$ such that $\beta = \text{mgu}((C_1, ..., C_n), (B_1, ..., B_n))$ and $A = K\beta$. The hypothesis c_1 subsumes c_2 implies that there exists a substitution σ such that $H\sigma = K$ and $\{A_1\sigma, ..., A_m\sigma\} \subseteq \{B_1, ..., B_n\}$. Therefore, for a suitable set of indexes $\{i_1, ..., i_m\} \subseteq \{1, ..., n\}$, there exists $\vartheta = \text{mgu}((C_{i_1}, ..., C_{i_m}), (A_1\sigma, ..., A_m\sigma))$. By Lemma A.7 (in the Appendix) there exists $\gamma = \text{mgu}((C_{i_1}, ..., C_{i_m}), (A_1, ..., A_m))$ and α such that $\gamma\alpha = \sigma\vartheta$. By definition of T_P^{ca}, $H\gamma \in T_Q^{ca}(X)$. Moreover $H\gamma \leqslant H\sigma\vartheta \leqslant H\sigma\beta = K\beta$, where the last inequality holds by definition of ϑ. This and the definition of $\prec$ complete the proof. ∎

COROLLARY 4.13. *Let P and Q be programs. Then*

1. *P and Q are subsumption equivalent iff $T_P^{ca} \prec T_Q^{ca}$ and $T_Q^{ca} \prec T_P^{ca}$,*

2. *P and Q are weakly-subsumption equivalent iff $T_P^{ca} + \text{id} \prec T_Q^{ca} + \text{id}$ and $T_Q^{ca} + \text{id} \prec T_P^{ca} + \text{id}$.*

Proof. For the *if* part of 1, if $T_P^{ca} \prec T_Q^{ca}$ and $T_Q^{ca} \prec T_P^{ca}$ then $T_P^s = T_Q^s$, since $T_P^s = \text{Ground}(T_P^{ca})$. Then, by Proposition 4.8, P and Q are subsumption equivalent. The *only if* is straightforward by Proposition 4.12.

The proof of 2 is similar to that in [30] for 2 of Proposition 4.8, by using Proposition 4.12. ∎

The following example shows why subsumption equivalence is not adequate when considering computed answers.

EXAMPLE 4.14. Let us consider programs $P = \{c_1\}$ and $Q = \{c_2\}$ where

$$c_1: q(X): -q(X). \qquad c_2: q(X): -q(X), q(X).$$

P and Q are subsumption equivalent but $T_Q^{ca} \neq T_P^{ca}$, since for $Z = \{q(f(a, X)) \cdot q(f(X, b))\}$ we have $q(f(a, b)) \in T_Q^{ca}(Z) \setminus T_P^{ca}(Z)$. Moreover programs $P \cup Z$ and $Q \cup Z$ are (weakly) subsumption equivalent but $P \cup Z \not\approx_{ca} Q \cup Z$, since the goal $q(X)$ can compute the answer $X/f(a, b)$ in $Q \cup Z$ only.

If clause c_1 subsumes clause c_2 we can only prove that for every atom A in $T_{\{c_2\}}^{ca}(Q)$ there exists a more general atom B in $T_{\{c_1\}}^{ca}(Q)$ and we cannot prove that there exists a variant of A in $T_{\{c_1\}}^{ca}(Q)$. Moreover, Example 4.14 shows that (weak) subsumption equivalence does not imply $\approx_{ca}$. Therefore, when considering computed answer substitutions, we cannot consider equivalent clauses which are subsumption equivalent: the logical meaning of subsumption is not adequate to represent the operational meaning of SLD derivation. This discrepancy is of the same nature of the weakness of the completeness theorem of SLD resolution (in general, only answers more general than correct answers can be computed [28]). In the program $Q \cup Z$ of Example 4.14, clause c_2 (which is a tautology) cannot be deleted nor replaced by c_1 preserving computed answer substitutions. If we are interested only in successful derivations, weak subsumption equivalence is perfectly adequate since in any refutation we can replace a clause c by a clause which subsumes c and we can delete a tautology without affecting the success of the derivation. Indeed, if P and Q are weak subsumption equivalent, then $T_P^s + \text{id} = T_Q^s + \text{id}$ and the programs have the same success set and the same successful derivations (Propositions 4.8 and 4.4).

5. COMPOSITIONAL EQUIVALENCES

We consider now equivalences obtained by considering $\cup_\Omega$ as composition operator. We first focus on computed answers as observable. A syntactic equivalence $\simeq$ on sets of clauses is used to define the equivalence $\sim_{(\Omega, ca)}$ and to obtain from the scheme the semantics $\mathscr{F}_{(\Omega, ca)}(P)$ which is correct wrt $\approx_{(\Omega, ca)}$. Full abstraction wrt this equivalence is then proved for the functional semantics $[\![P]\!]^{ca}$. Finally we take into account successful derivations: by using an equivalence $\sim_{(\Omega, s)}$ based on weak subsumption equivalence, we obtain the semantics $\mathscr{F}_{(\Omega, s)}(P)$ which is fully abstract wrt $\approx_{(\Omega, s)}$.

5.1. *A Syntactic Equivalence on Clauses*

We want a syntactic equivalence $\simeq$ on (sets of) clauses which preserves their operational meaning in the case of computed answers. A distinction can be made among the atoms in the body of a clause, by identifying those "relevant" atoms which can share variables with the head in a derivation, and those which cannot. Clearly, only the atoms of the first type can contribute to the answer computed in a derivation. The others can only be "tested" for their successful derivation, but their derivation cannot give any useful binding for the computed answer, since such an answer is always restricted to the variables in the goal. According to our discussion in the previous section, we can consider subsumption on the atoms of the second type and not on relevant atoms. Hence the following.

DEFINITION 5.1. An atom B in the body of a clause c is called *relevant* if either it shares variables with the head of c or, inductively, it shares variables with another atom B' in the body of c which is relevant. The *multiset* of relevant atoms in c is denoted by $\text{Rel}(c)$.

In the following $\text{Set}(M)$ denotes the set of the elements which appear in the multiset (or sequence) M. Moreover, when applied to multisets, $\subseteq$ denotes multiset inclusion defined as $\{|A_1, ..., A_n|\} \subseteq \{|B_1, ..., B_m|\}$ iff there exists a set of (distinct) indexes $\{j_1, ..., j_n\} \subseteq \{1, ..., m\}$ such that $\{|A_1, ..., A_n|\} = \{|B_{j_1}, ..., B_{j_n}|\}$. We first define a pre-order on clauses.

DEFINITION 5.2. Let c_1 and c_2 be two clauses which do not share variables and whose heads are H_1 and H_2, respectively. We define $c_1 \leqslant_c c_2$ iff c_1 subsumes c_2 and there exists a renaming ρ such that $H_1\rho = H_2$, $\mathrm{Rel}(c_2) \subseteq \mathrm{Rel}(c_1\rho)$ and $\mathrm{Set}(\mathrm{Rel}(c_2)) = \mathrm{Set}(\mathrm{Rel}(c_1\rho))$.

We define then the equivalence $\simeq$ as the symmetric closure of the Smith preordering induced on sets of clauses by $\leqslant_c$.

DEFINITION 5.3. Let P, Q be set of clauses. We define $P \lhd Q$ iff for any $c_2 \in Q$ there exists $c_1 \in P$ such that $c_1 \leqslant_c c_2$. Moreover, we define $P \simeq Q$ iff $P \lhd Q$ and $Q \lhd P$.

In the previous definitions relevant atoms in clause bodies are considered as multisets rather than sets.

This is because, as shown by Example 4.14, in general a relevant atom in the body $\tilde{B}$ of a clause cannot be deleted (even if a copy of the atom appear in $\tilde{B}$) without changing the operational meaning of the clause in terms of computed answers. By making reference to Example 4.14 we then have $c_2 \leqslant_c c_1$ while $c_1 \not\leqslant_c c_2$. Clearly, also relevant atoms which contain local variables only cannot be deleted (see Example 5.4).

By Theorem 5.5, the equivalence $\simeq$ implies the T^{ca}_P's equality but the converse does not hold. Moreover (Corollary 5.6), if $P \lhd Q$ then P behaves as Q under any composition, while if $P \simeq Q$ then the two sets of clauses are indistinguishable by $\approx_{(\Omega, \mathrm{ca})}$. As a consequence, $\simeq$ can be used to correctly define the equivalence $\sim_{(\Omega, \mathrm{ca})}$ (see the next subsection).

EXAMPLE 5.4. Consider the programs

$$P = \{\, p(X) \colon -q(X, Y), r(Y).\}$$

$$Q = \{\, p(X) \colon -q(X, Y), r(Y), r(Y).\}$$

$$R = \{\, q(X, X).$$
$$\quad r(f(a, Y)).$$
$$\quad r(f(Y, b)).\ \}$$

The goal $p(X, Y)$ has the computed answer $\vartheta = \{X/f(a, b)\}$ in $Q \cup R$ and not in $P \cup R$. Therefore $P \cup R \not\approx_{\mathrm{ca}} Q \cup R$

THEOREM 5.5. *Let P, Q be programs. If $P \lhd Q$ then, for any X, $T^{ca}_Q(X) \subseteq T^{ca}_P(X)$ and the converse does not hold.*

Proof. First observe that, according to the definition of T^{ca}_P, if P' is a renamed version of P then, for any I, $T^{ca}_P(I) = T^{ca}_{P'}(I)$. Moreover, the order of the atoms in the bodies of clauses in P is not relevant to the result of $T^{ca}_P(I)$.

Let us suppose that $A \in T^{ca}_Q(X)$ for $X \subseteq \mathscr{A}$. By definition of $T^{ca}_Q(X)$, there exists a clause c_2: $H_2 \colon -\tilde{B}_2, \tilde{R}_2 \in Q$, and there exists a sequence $\tilde{B}$, $\tilde{C}$ of renamed apart atoms in X such that

$$\text{there exists } \vartheta = \mathrm{mgu}((\tilde{B}_2, \tilde{R}_2), (\tilde{B}, \tilde{C})) \tag{1}$$

and $A = H_2\vartheta$. According to the previous remark on the order of the atoms in the bodies, we can assume without loss of generality that $\tilde{R}_2$ is the sequence of all the relevant atoms in c_2.

Since, by hypothesis, $P \lhd Q$, there exists a clause c_1: $H_1 \colon -\tilde{B}_1, \tilde{R}_1 \in P$ such that $c_1 \leqslant_c c_2$ (also in this case we assume that $\tilde{R}_1$ contains the relevant atoms). Let ρ be a renaming as in Definition 5.2. Then $c_1\rho$ subsumes c_2, since $c_1 \leqslant_c c_2$. Proposition 4.12 and the definition of $\prec$ imply that there exists a sequence $\tilde{D}$, $\tilde{D}'$ of renamed apart atoms in X such that

$$\text{there exists } \beta = \mathrm{mgu}((\tilde{B}_1\rho, \tilde{R}_1\rho), (\tilde{D}, \tilde{D}')) \tag{2}$$

(and $H_1\beta \leqslant A$).

The hypothesis $c_1 \leqslant_c c_2$ implies $\{|\tilde{R}_2|\} \subseteq \{|\tilde{R}_1\rho|\}$. According to the previous remark on the order of atoms in the bodies, we can assume without loss of generality that $\tilde{R}_2$, $\tilde{R}_+ = \tilde{R}_1\rho$. Moreover, since the definition of $\leqslant_c$ implies also that $\mathrm{Set}(\tilde{R}_2) = \mathrm{Set}(\tilde{R}_1\rho)$, any atom in $\tilde{R}_+$ appears in $\tilde{R}_2$. More precisely, if $\tilde{R}_2$ denotes the sequence $R_1, \ldots, R_m$, we have that

$$\tilde{R}_2, R_{j_1}, \ldots, R_{j_n} = \tilde{R}_1\rho, \tag{3}$$

where $1 \leqslant j_i \leqslant m$ for all i, $1 \leqslant i \leqslant n$.

Now (1), (3), and a straightforward inspection of the unification algorithm imply that

$$\vartheta = \mathrm{mgu}((\tilde{B}_2, \tilde{R}_1\rho), (\tilde{B}, \tilde{C}, \tilde{C}_+)) \tag{4}$$

where, if $\tilde{C}$ denotes the sequence $C_1, \ldots, C_m$ then $\tilde{C}_+ = C_{j_1}, \ldots, C_{j_n}$ with the indexes defined as before.

By definition of relevant atom $\tilde{R}_1\rho$ and $\tilde{B}_1\rho$ do not share any variable. Since we can assume that $\tilde{D}$ is a sequence of renamed apart atoms, (2), (4), and the definition of the unification algorithm imply that

$$\text{there exists } \gamma = \mathrm{mgu}((\tilde{B}_1\rho, \tilde{R}_1\rho), (\tilde{D}, \tilde{C}, \tilde{C}_+)) \text{ such that } \vartheta_{|\mathrm{Var}(\tilde{R}_2)} = \gamma_{|\mathrm{Var}(\tilde{R}_1\rho)}. \tag{5}$$

Therefore $A = H_2\vartheta = H_1\rho\gamma$. Moreover, since by construction $\tilde{D}$, $\tilde{C}$, $\tilde{C}_+$ is a sequence of atoms in X, $H_1\rho\gamma \in T^{ca}_{P\rho}(X)$. The mentioned equality $T^{ca}_{P\rho}(X) = T^{ca}_P(X)$ concludes the proof of the implication in the thesis. A counter example for the converse is given by programs P and Q of Example 5.14. ∎

COROLLARY 5.6. *Let P, Q, R be programs such that $P \lhd Q$. If ϑ is a computed answer for the goal G in $Q \cup R$ then ϑ is a computed answer for G in $P \cup R$. Moreover if $P \simeq Q$ then $P \approx_{(\Omega, \mathrm{ca})} Q$ for any Ω.*

Proof. By the strong completeness theorem for the s-semantics in [17], ϑ is a computed answer for the goal

G in P iff ϑ is a computed answer for G in (the program obtained from) $\mathscr{F}_{ca}(P)$. Since by definition $\mathscr{F}_{ca}(P) = T_P^{ca} \uparrow \omega$ and T_P^{ca} is monotonic wrt $\subseteq$, if $P \lhd Q$ then, by Proposition 5.5, $\mathscr{F}_{ca}(Q) \subseteq \mathscr{F}_{ca}(P)$. In order to complete the proof of the first part, we only need to observe that $P \lhd Q$ implies $P \cup R \lhd Q \cup R$ for any R. The second part is a straightforward consequence of the first one and of Definitions 2.4 and 5.3. ▮

5.2. *A Semantics Correct wrt* $\approx_{(\Omega, ca)}$

By using the previous equivalence $\simeq$ we obtain now the instance of the scheme $\mathscr{F}_{(\Omega, ca)}(P)$. This semantics is compositional wrt $\cup_\Omega$ and correctly models computed answers, i.e., it is correct wrt $\approx_{(\Omega, ca)}$. A semantics with these features was already defined in [8] by using sets of clauses as interpretations. Our interest here is to show how such a semantics can be obtained from the general scheme. Moreover, [8] uses identity as $\sim$ while we use a coarser equivalence. This allows us to obtain a more (even if not fully) abstract semantics.

By Corollary 5.6 we can derive that $\simeq$ equivalent clauses can be interchanged in any context while preserving the computed answer substitutions semantics. We can then use $\simeq$ to define the equivalence $\sim_{(\Omega, ca)}$. Moreover, since $\cup_\Omega$ allows us to compose programs which share predicate symbols in Ω only, we only need the information given by clauses in $\mathscr{C}^\Omega$ (Definition 3.1). Therefore we have the following.

DEFINITION 5.7. Let $I, J \subseteq \mathscr{C}$. We define $I \sim_{(\Omega, ca)} J$ iff $I \cap \mathscr{C}^\Omega \simeq J \cap \mathscr{C}^\Omega$ where $\simeq$ is defined in Definition 5.3.

As shown below, $\sim_{(\Omega, ca)}$ is finer than (and hence correct wrt) $\approx_{(\Omega, ca)}$.

LEMMA 5.8. *Let* $I, J \subseteq \mathscr{C}^\Omega$. *If* $I \sim_{(\Omega, ca)} J$ *then* $I \approx_{(\Omega, ca)} J$.

Proof. The proof is straightforward by the previous definition, Corollary 5.6 and the definition of $\approx_{(\Omega, ca)}$. ▮

The usual requirements on $\sim$ are guaranteed by the following.

PROPOSITION 5.9. *Let* $I, J \subseteq \mathscr{C}$. *The following properties hold*:

(i) $\sim_{(\Omega, ca)}$ *is a congruence* wrt *infinite unions*,

(ii) *if* $I \sim_{(\Omega, ca)} J$ *then* $\Gamma_P(I) \sim_{(\Omega, ca)} \Gamma_P(J)$.

Note that for $\Omega = \varnothing$, $\sim_{(\Omega, ca)}$ is the same as $\sim_{ca}$ of Section 4. The semantics $\mathscr{F}_{(\Omega, ca)}(P)$ is obtained from the scheme as follows.

DEFINITION 5.10 (Compositional Computed Answers Semantics). Let P be a program. The semantics $\mathscr{F}_{(\Omega, ca)}(P)$ is defined as the instance of Definition 3.10 obtained by using $\sim_{(\Omega, ca)}$ as $\sim$.

Theorem 5.12 and Corollary 5.13 show the correctness properties for this semantics. Essentially the same results have already been given in [9, 8] by using the identity on $\wp(\mathscr{C})$ as $\sim_{(\Omega, ca)}$ equivalence.

LEMMA 5.11. *Let* P *be a program such that* $P \subseteq \mathscr{C}^\Omega$ *and let us denote by* Q *the program* $\iota([P]) \cap \mathscr{C}^\Omega$. *Then* $\Gamma_P(I) \sim_{(\Omega, ca)} \Gamma_Q(I)$.

Proof. The proof is similar to the one of the first part of Proposition 5.9. ▮

THEOREM 5.12. *Let* P, Q *be* Ω*-open programs such that* $P \cup_\Omega Q$ *is defined. Then*

1. $\mathscr{F}_{(\Omega, ca)}(\iota(\mathscr{F}_{(\Omega, ca)}(P) \sqcup \mathscr{F}_{(\Omega, ca)}(Q)) \cap \mathscr{C}^\Omega) = \mathscr{F}_{(\Omega, ca)}(P \cup_\Omega Q)$,
2. $P \approx_{(\Omega, ca)} \iota(\mathscr{F}_{(\Omega, ca)}(P)) \cap \mathscr{C}^\Omega$.

Proof. In the following $[I]$ denotes a $\sim_{(\Omega, ca)}$ equivalence class. Let us define $S(P) = \mathscr{F}(P) \cap \mathscr{C}^\Omega$. Under the previous hypothesis, Theorems 2.8 and 2.9 in [8] can be stated as follows

$$S(S(P) \cup S(Q)) = S(P \cup_\Omega Q) \tag{1}$$

$$P \approx_{(\Omega, ca)} S(P). \tag{2}$$

By Definition 5.7,

$$\mathscr{F}_{(\Omega, ca)}(P) = [S(P)]. \tag{3}$$

Moreover, by Definition 3.4 and by Definition 5.7,

$$\mathscr{F}_{(\Omega, ca)}(P) \sqcup \mathscr{F}_{(\Omega, ca)}(Q) = [S(P) \cup S(Q)]. \tag{4}$$

Finally, by Lemma 5.11 and since $S(P) \subseteq \mathscr{C}^\Omega$,

$$\mathscr{F}_{(\Omega, ca)}(\iota([S(P)]) \cap \mathscr{C}^\Omega) = \mathscr{F}_{(\Omega, ca)}(S(P)). \tag{5}$$

Therefore, we have the following equalities

$$\begin{aligned}
&\mathscr{F}_{(\Omega, ca)}(\iota(\mathscr{F}_{(\Omega, ca)}(P) \sqcup \mathscr{F}_{(\Omega, ca)}(Q)) \cap \mathscr{C}^\Omega) && \text{(by 4)}\\
&= \mathscr{F}_{(\Omega, ca)}(\iota([S(P) \cup S(Q)]) \cap \mathscr{C}^\Omega) && \text{(by 5)}\\
&= \mathscr{F}_{(\Omega, ca)}(S(P) \cup S(Q)) && \text{(by 3)}\\
&= [S(S(P) \cup S(Q))] && \text{(by 1)}\\
&= [S(P \cup_\Omega Q)] && \text{(by 3)}\\
&= \mathscr{F}_{(\Omega, ca)}(P \cup_\Omega Q)
\end{aligned}$$

and part 1 of the thesis holds. Part 2 is a straightforward consequence of (2) and of Lemma 5.8. ▮

COROLLARY 5.13. *Let* P, Q *be programs. If* $\mathscr{F}_{(\Omega, ca)}(P) = \mathscr{F}_{(\Omega, ca)}(Q)$ *then* $P \approx_{(\Omega, ca)} Q$.

The converse of the corollary does not hold, i.e., the semantics $\mathscr{F}_{(\Omega, \mathrm{ca})}(P)$ is not fully abstract wrt $\approx_{(\Omega, \mathrm{ca})}$ (see Example 5.14). The difficulty here is related to the use of clauses in the semantic domain (the full abstraction result in [24] was obtained using a domain not containing clauses).

We can observe that, as shown by Example 5.14, $P \approx_{(\Omega, \mathrm{ca})} Q$ does not imply that there exists $c_1 \in P$ and $c_2 \in Q$ such that $\{c_1\} \approx_{(\Omega, \mathrm{ca})} \{c_2\}$. This means that any semantic domain which contains equivalences classes of clauses (instead of equivalence classes of sets of clauses) does not achieve full abstraction.

EXAMPLE 5.14. Let us consider the programs

$$P = \{c_1: p(x): -q(x, b).$$
$$c_2: p(x): -q(x, y).\}$$
$$Q = \{c_2: p(x): -q(x, y).$$
$$c_3: p(x): -q(x, b), q(y, a).\}$$

It is easy to show that, for $\Omega = \{q\}$, $\{c_1\} \not\approx_{(\Omega, \mathrm{ca})} \{c_2\}$, $\{c_1\} \not\approx_{(\Omega, \mathrm{ca})} \{c_3\}$ and $\{c_2\} \not\approx_{(\Omega, \mathrm{ca})} \{c_3\}$.

However $T_P^{\mathrm{ca}} = T_Q^{\mathrm{ca}}$ and $P \approx_{(\Omega, \mathrm{ca})} Q$, since every answer in any $\cup_\Omega$ context that can be computed by using c_1 can be computed by using either c_2 or c_3, and any answer computed by c_3 can be computed by c_1. Note that, according to Definitions 5.2 and 5.3, $c_1 \leqslant_c c_3$, $c_3 \not\leqslant_c c_1$, $c_2 \not\leqslant_c c_1$. Therefore $P \not\approx Q$ and $\mathscr{F}_{(\Omega, \mathrm{ca})}(P) \neq \mathscr{F}_{(\Omega, \mathrm{ca})}(Q)$.

5.3. *A Semantics Fully Abstract with Respect to* $\approx_{(\Omega, \mathrm{ca})}$

We give now a result (Theorem 5.17) which identifies the equivalence $\approx_{(\Omega, \mathrm{ca})}$ with the equivalence induced by the equality of the functions $[\![P]\!]^{\mathrm{ca}}$, in the case of a domain restricted to $\wp(\mathscr{A}^\Omega)$. In other words, the functional semantics given by such a restriction of $[\![P]\!]^{\mathrm{ca}}$ is fully abstract wrt $\approx_{(\Omega, \mathrm{ca})}$.

We first need the following theorem which shows that, when considering computed answers, two programs can be distinguished by $\cup_\Omega$ composition iff they can be distinguished by additions of atoms whose predicate symbols are in Ω. This generalizes the result given in [24] for the case $\Omega = \Pi$.

THEOREM 5.15. *Let P, Q be programs. Then $P \not\approx_{(\Omega, \mathrm{ca})} Q$ iff there exists a set of atoms $X \subseteq \mathscr{A}^\Omega$ such that $P \cup X \not\approx_{\mathrm{ca}} Q \cup X$.*

LEMMA 5.16. *Let P be a program and Q be a set of atoms. Then $T^{\mathrm{ca}}_{P \cup Q} \uparrow \omega = (T_P^{\mathrm{ca}} + \mathrm{id})^\omega(Q)$.*

THEOREM 5.17 (Full abstraction). *Let P and Q be programs. For any Ω, $P \approx_{(\Omega, \mathrm{ca})} Q$ iff $\forall X \subseteq \mathscr{A}^\Omega$, $[\![P]\!](X) = [\![Q]\!](X)$.*

Proof. We have the following equivalences:

$P \approx_{(\Omega, \mathrm{ca})} Q$
iff (by Definition 2.4 and Theorem 5.15)
$\forall X \subseteq \mathscr{A}^\Omega$, $P \cup X \approx_{\mathrm{ca}} Q \cup X$
iff (by Theorem 4.6)
$\forall X \subseteq \mathscr{A}^\Omega$, $\mathscr{F}_{\mathrm{ca}}(P \cup X) = \mathscr{F}_{\mathrm{ca}}(Q \cup X)$
iff (by Definition 3.10)
$\forall X \subseteq \mathscr{A}^\Omega$, $T^{\mathrm{ca}}_{P \cup X} \uparrow \omega = T^{\mathrm{ca}}_{Q \cup X} \uparrow \omega$
iff (by Lemma 5.16)
$\forall X \subseteq \mathscr{A}^\Omega$, $(T_P^{\mathrm{ca}} + \mathrm{id})^\omega(X) = (T_Q^{\mathrm{ca}} + \mathrm{id})^\omega(X)$
iff (by Definition of $[\![\ ]\!]^{\mathrm{ca}}$
$\forall X \subseteq \mathscr{A}^\Omega$, $[\![P]\!]^{\mathrm{ca}}(X) = [\![Q]\!]^{\mathrm{ca}}(X)$

and this completes the proof. ∎

For $\Omega = \Pi$ we have the following special case.

COROLLARY 5.18. *Let P and Q be programs. Then $P \approx_{(\Pi, \mathrm{ca})} Q$ iff $[\![P]\!]^{\mathrm{ca}} = [\![Q]\!]^{\mathrm{ca}}$.*

The restriction to $\wp(\mathscr{A}^\Omega)$ in Theorem 5.17 is necessary since, as shown by Example 5.19, the equivalence induced by $[\![P]\!]^{\mathrm{ca}}$ is strictly finer than the equivalence induced by the restriction of $[\![P]\!]^{\mathrm{ca}}$ to $\wp(\mathscr{A}^\Omega)$.

EXAMPLE 5.19. Let $P = \{p(a): -q(a).\}$ and $Q = \varnothing$. If $\Omega = \varnothing$ then $[\![P]\!]^{\mathrm{ca}}$ and $[\![Q]\!]^{\mathrm{ca}}$ are equal when restricted to $\wp(\mathscr{A}^\Omega) = \varnothing$. However, if $\Omega = \{q\}$ then there exists $X = \{q(a)\} \in \wp(\mathscr{A}^\Omega)$ such that $[\![P]\!]^{\mathrm{ca}}(X) \neq [\![Q]\!]^{\mathrm{ca}}(X)$.

For larger sets of open predicates we obtain finer $\approx_{(\Omega, \mathrm{ca})}$ equivalences. The following proposition formally states this property.

PROPOSITION 5.20. *Let P and Q be programs and let $\Psi \subseteq \Omega$. If $P \approx_{(\Omega, \mathrm{ca})} Q$ then $P \approx_{(\Psi, \mathrm{ca})} Q$ and the converse does not hold.*

Proof. The hypothesis implies that $\mathscr{A}^\Psi \subseteq \mathscr{A}^\Omega$. Then we only need to apply Definition 2.4 and Theorem 5.15. For the converse consider Example 5.19 with $\Psi = \varnothing$ and $\Omega = \{q\}$. ∎

5.4. *The Equivalence* $\approx_{(\Omega, s)}$

In this subsection we consider the usual program composition $\cup_\Omega$ but we will focus on successful derivations as observable. We will obtain from the general scheme a semantics $\mathscr{F}_{(\Omega, s)}(P)$ which is fully abstract wrt $\approx_{(\Omega, s)}$.

According to the general construction, we have only to define a suitable equivalence $\sim_{(\Omega, s)}$ on clauses. Since here

we are concerned only with successful derivations, $\sim_{(\Omega, s)}$ can simply be defined in terms of weak subsumption equivalence. Indeed, as previously discussed, if a clause c subsumes c' then each successful derivation of G can be performed by using c instead of c'. Moreover, note that the clause c is a tautology iff the body of c contains a copy of the head. Therefore, if G has a successful derivation which uses the tautology c, G has also a derivation which does not use c. In other words, tautological clauses can be deleted. These remarks can be formalized as follows.

DEFINITION 5.21. Let $I, J \subseteq \mathscr{C}$. We define $I \sim_{(\Omega, s)} J$ iff $I \cap \mathscr{C}^{\Omega}$ is weakly subsumption equivalent to $J \cap \mathscr{C}^{\Omega}$.

The equivalence $\sim_{(\Omega, \mathrm{ca})}$ is strictly finer than $\sim_{(\Omega, s)}$, since if $I \sim_{(\Omega, \mathrm{ca})} J$ then $I \cap \mathscr{C}^{\Omega}$ and $J \cap \mathscr{C}^{\Omega}$ are subsumption equivalent. The following is the usual result.

PROPOSITION 5.22. *Let $I, J \subseteq \mathscr{C}$. The following properties hold:*

(i) $\sim_{(\Omega, s)}$ *is a congruence wrt infinite unions,*

(ii) *if* $I \sim_{(\Omega, s)} J$ *then* $\Gamma_P(I) \sim_{(\Omega, s)} \Gamma_P(J)$.

DEFINITION 5.23 (Compositional Successful Derivation Semantics). Let P be a program. The semantics $\mathscr{F}_{(\Omega, s)}(P)$ is defined as the instance of Definition 3.10 obtained by using $\sim_{(\Omega, s)}$ as $\sim$.

Before proving the full abstraction of the semantics, we need some preliminary facts which are the ground case counterparts of previously stated results.

PROPOSITION 5.24. *Let P and Q be programs. Then $P \not\approx_{(\Omega, s)} Q$ iff there exists a set of ground atoms $R \subseteq \mathscr{A}^{\Omega}$ such that $P \cup R \not\approx_s Q \cup R$.*

Proof. The proof is similar to the one of Theorem 5.15. ∎

PROPOSITION 5.25 [26]. *Let P be a program. For any ground set of atoms Q, $[\![P]\!]^s(Q) = T^s_{P \cup Q} \uparrow \omega$.*

PROPOSITION 5.26. *Let P and Q be programs and Ω be a set of predicate symbols. $P \approx_{(\Omega, s)} Q$ iff, for any set of ground atoms $R \subseteq \mathscr{A}^{\Omega}$, $[\![P]\!]^s(R) = [\![Q]\!]^s(R)$.*

Proof. We have the following equivalences (where R is a ground subset of $\mathscr{A}^{\Omega}$)

$P \approx_{(\Omega, s)} Q$

iff (by Definition 2.4 and Proposition 5.24)

$\forall R,\ P \cup R \approx_s Q \cup R$

iff (by Proposition 4.4)

$\forall R,\ M(P \cup R) = M(Q \cup R)$

iff (by standard results)

$\forall R,\ T^s_{P \cup R} \uparrow \omega = T^s_{Q \cup R} \uparrow \omega$

iff (by Lemma 5.25)

$\forall R,\ [\![P]\!]^s(R) = [\![Q]\!]^s(R)$

and this completes the proof. ∎

In the special case $\Omega = \Pi$, the previous result can be obtained in a simpler way by using part 4 of Proposition 4.8, Proposition 4.4, and the fact that P and Q are logically equivalent iff they have the same set of atomic logic consequences [23].

DEFINITION 5.27. Let P be a program. P is u-closed iff $\mathrm{unf}_{P \cup \mathrm{Id}_{\Pi}}(P) = P$.

LEMMA 5.28 [14]. *Let P, Q, R be programs. Then* $\mathrm{unf}_R(\mathrm{unf}_Q(P)) = \mathrm{unf}_{\mathrm{unf}_R(Q)}(P)$.

LEMMA 5.29. *If P is a u-closed program then, for any set of ground atoms Q, $[\![P]\!]^s(Q) = (T^s_P + \mathrm{id})(Q)$.*

Proof. Let us denote $P \cup \mathrm{Id}_{\Pi}$ by P^* and Ground $(\mathrm{unf}_Q(P))$ by $\mathrm{unf}\, g_Q(P)$. By definition of T^s_P, $(T^s_P + \mathrm{id})(Q) = \mathrm{unf}\, g_Q(P^*)$. Then we have the following equalities where Q denotes a set of ground atoms:

$$
\begin{aligned}
&(T^s_P + \mathrm{id})((T^s_P + \mathrm{id})(Q)) \\
&\quad = \mathrm{unf}\, g_{\mathrm{unf}\, g_Q(P^*)}(P^*) && \text{(by the previous equality)} \\
&\quad = \mathrm{unf}\, g_Q(\mathrm{unf}\, g_{P^*}(P^*)) && \text{(by Lemma 5.28)} \\
&\quad = \mathrm{unf}\, g_Q(P^*) && \text{(since } P \text{ is } u\text{-closed)} \\
&\quad = (T^s_P + \mathrm{id})(Q) && \text{(by the previous equality).}
\end{aligned}
$$

Then the thesis follows by a straightforward inductive argument. ∎

We can give now the following result. A fully abstract invariant wrt logical equivalence (and therefore wrt $\approx_{(\Pi, s)}$) was already given in [23].

LEMMA 5.30 [8]. *Let P be a set of clauses and Ω be a set of predicate symbols. Then $\mathscr{F}(P) \cap \mathscr{C}^{\Omega}$ is u-closed.*

THEOREM 5.31 (Full Abstraction). *Let P, Q be (finite) programs. Then $P \approx_{(\Omega, s)} Q$ iff $\mathscr{F}_{(\Omega, s)}(P) = \mathscr{F}_{(\Omega, s)}(Q)$.*

Proof. In order to simplify the notation let us denote $\mathscr{F}(P) \cap \mathscr{C}^{\Omega}$ and $\mathscr{F}(Q) \cap \mathscr{C}^{\Omega}$ by $S(P)$ and $S(Q)$, respectively.

By a result of [8] (reported in the Proof of Theorem 5.12) $P \approx_{(\Omega, s)} S(P)$. Then $P \approx_{(\Omega, s)} Q$ iff $S(P) \approx_{(\Omega, s)} S(Q)$ and, to prove the thesis, we have only to show that $S(P) \approx_{(\Omega, s)} S(Q)$ iff $S(P)$ and $S(Q)$ are weak subsumption equivalent (w.s.e.). By the previous lemma $S(P)$ and $S(Q)$ are u-closed. Then we have the following implications where R denotes a ground subset of $\mathscr{A}^{\Omega}$

$S(P) \approx_{(\Omega, s)} S(Q)$

iff (by Proposition 5.26)

$\forall R, [\![S(P)]\!]^s(R) = [\![S(Q)]\!]^s(R)$

iff (by Lemma 5.29)

$\forall R, (T^s_{S(P)} + \mathrm{id})(R) = (T^s_{S(Q)} + \mathrm{id})(R)$

iff (by 2 of Proposition 4.8)

$S(P)$ w.s.e. $S(Q)$.

Note that property 2 of Proposition 4.8 is stated for finite programs. However, as resulting from its proof in [30], it can be stated also for infinite programs which contain only a finite number of function symbols (the proof needs new constant symbols which do not appear in the programs). Since the programs P and Q are finite, $S(P)$ and $S(Q)$ contain only a finite number of function symbols. Therefore the last *iff* holds and this completes the proof. ∎

Note that the previous result holds also for infinite programs which contain only finitely many function symbols. It does not hold for generic infinite programs (for a counterexample consider the programs P and Ground(P)).

6. A SEMANTICS FOR PARTIAL ANSWERS AND CALL PATTERNS

A fixpoint semantics for partial answers has already been defined in [16]. We extend here such a characterization by obtaining, from the general scheme, a fully abstract semantics for partial answers and a correct semantics for correct partial answers. Semantics for call patterns will also be given.

As previously discussed, the "collecting semantics" $\mathscr{F}(P)$ contains all the useful information on any SLD derivation (recall that in $\mathscr{F}(P)$ the equivalence $\sim$ defining the semantic domain is the identity). As shown by Lemmata 6.5 and 6.12, we can then extract from the clauses in $\mathscr{F}(P)$ also the information needed to model partial answers and call patterns for any goal G. For example, since each clause $H\colon -\tilde{B}$ in $\mathscr{F}(P)$ corresponds to a derivation $p(\tilde{X}) \overset{\beta}{\rightarrow}_{P,R} \tilde{B}$ where $H = p(\tilde{X})\,\beta$, ϑ is a partial answer for the goal $p(\tilde{X})$ if there exists a clause $H\colon -\tilde{B}$ in $\mathscr{F}(P)$ such that $\vartheta = \mathrm{mgu}(p(\tilde{X}), H)$. Moreover, ϑ is a correct partial answer for $p(\tilde{t})$ if there exists also a conjunction $\tilde{C}$ containing atoms from $\mathscr{F}(P)$ such that $\tilde{B}$ and $\tilde{C}$ unify. Lemma 6.5 shows the extension to the general case of the argument used in this example.

Note that, when considering partial answers, we only need the information in the heads of the clauses in $\mathscr{F}(P)$, while for correct partial answers clearly we have to consider also bodies. In fact, bodies contain the information needed to check if the partial derivation is part of a refutation. For example, if $P = \{p(a)\colon -q(a).\}$ then $\mathscr{F}(P) = [P]$ and X/a is the mgu of $p(X)$ with the head of a clause c in $\mathscr{F}(P)$. However, X/a is not a correct partial answer, because the body of c does not unify with any unit clause in $\mathscr{F}(P)$. According to these considerations we can then define the equivalences $\sim_{\mathrm{pa}}$ and $\sim_{\mathrm{cpa}}$ as follows.

DEFINITION 6.1. Assume that $J \subseteq \mathscr{C}$. Then we define

$$\mathrm{Heads}(J) = \{H \in \mathscr{A} \mid H\colon -\tilde{B} \in J\}.$$

DEFINITION 6.2. Let P be a program and let $I, J \subseteq \mathscr{C}$. We define $I \sim_{\mathrm{pa}} J$ iff $\mathrm{Heads}(I \cup \mathrm{Id}_H) = \mathrm{Heads}(J \cup \mathrm{Id}_H)$. Moreover, we define $I \sim_{\mathrm{cpa}} J$ iff $I \sim_{(H,\mathrm{ca})} J$.

PROPOSITION 6.3. *Let $I, J \subseteq \mathscr{C}$. The following properties hold*

(i) $\sim_{\mathrm{pa}}$ *is a congruence wrt infinite unions,*

(ii) *if* $I \sim_{\mathrm{pa}} J$ *then* $\Gamma_P(I) \sim_{\mathrm{pa}} \Gamma_P(J)$.

The semantics for (correct) partial answers can be obtained as usual from the general scheme.

DEFINITION 6.4 ((Correct) Partial Answers Semantics). Let P be a program. The semantics $\mathscr{F}_{\mathrm{pa}}(P)$ for partial answers and $\mathscr{F}_{\mathrm{cpa}}(P)$ for correct partial answers are defined as the instances of Definition 3.10 obtained by using $\sim_{\mathrm{pa}}$ and $\sim_{\mathrm{cpa}}$, respectively.

The following lemma shows how we can obtain a partial answer (correct partial answer) for a goal G from $\mathscr{F}_{\mathrm{pa}}(P)$ (from $\mathscr{F}_{\mathrm{cpa}}(P)$).

LEMMA 6.5. *Let P be a program and G be a goal.*

1. *ϑ is a partial answer for G in P iff there exists a subgoal G' of G and there exists a sequence $\tilde{H}$ of heads of clauses in $\iota(\mathscr{F}_{\mathrm{pa}}(P))$ such that $\gamma = \mathrm{mgu}(G', \tilde{H})$ and $\vartheta = \gamma_{|\mathrm{Var}(G)}$,*

2. *ϑ is a correct partial answer for G in P iff there exists a subgoal $G' = A_1, \ldots, A_n$ of G and there exists a sequence $\tilde{H}$ of heads of n clauses $H_i\colon -\tilde{B}_i$ in $\iota(\mathscr{F}_{\mathrm{cpa}}(P))$ $(i = 1, \ldots, n)$ such that*

   (a) $\gamma = \mathrm{mgu}(G', \tilde{H})$ *and* $\vartheta = \gamma_{|\mathrm{Var}(G)}$,

   (b) *there exists a sequence $\tilde{C}$ of atoms in $\iota(\mathscr{F}_{\mathrm{cpa}}(P))$ such that $G''\gamma$ and $\tilde{C}$ are unifiable where G'' denotes the goal obtained from G by replacing each A_i by $\tilde{B}_i$.*

Proof. If ϑ is the empty partial answer then we can consider the empty sequence as $\tilde{H}$. Otherwise, by definition, ϑ is a partial answer for G in P iff there exists a derivation $G \overset{\beta}{\rightarrow}_P \tilde{A}$ with $\vartheta = \beta_{|\mathrm{Var}(G)}$. Moreover, ϑ is a correct partial answer iff there exists also a refutation $\tilde{A} \overset{\delta}{\rightarrow}_P \square$.

By Proposition 3.12, $G \overset{\beta}{\rightarrow}_P \tilde{A}$ iff there exist G' subgoal of G, n clauses $H_i\colon -\tilde{B}_i$ in $\mathscr{F}(P)$, and there exists G'' as specified in the hypothesis such that

$$\gamma = \mathrm{mgu}(G', \tilde{H}) \quad \text{and} \quad \beta_{|\mathrm{Var}(G)} = \gamma_{|\mathrm{Var}(G)}, \tag{1}$$

$$\{|\tilde{A}|\} = \{|G''\gamma|\}, \tag{2}$$

where $\tilde{H}$ is the sequence of the heads of the n clauses. By Definition 6.2, $H: -\tilde{B} \in \iota(\mathscr{F}(P))$ iff H is the head of a (variance equivalence class of) clause(s) in $\iota(\mathscr{F}_{\mathrm{pa}}(P))$ and in $\iota(\mathscr{F}_{\mathrm{cpa}}(P))$. Then by (1) the thesis in part 1 holds. In order to complete the proof of part 2 we can observe that $\tilde{A}$ has a refutation in P iff (by the switching lemma [28] and by (2)) $G''\gamma$ has a refutation in P iff (by Proposition 3.12) there exists a sequence $\tilde{C}$ of (renamed apart elements of variance equivalence classes of) atoms in $\iota(\mathscr{F}(P))$ such that $G''\gamma$ and $\tilde{C}$ are unifiable. Since, by Definition 6.2, C is a (variant equivalence class of) atom(s) in $\iota(\mathscr{F}(P))$ iff C is in $\iota(\mathscr{F}_{\mathrm{cpa}}(P))$, this completes the proof. ∎

The following theorem shows that $\mathscr{F}_{\mathrm{pa}}(P)$ is fully abstract wrt $\approx_{\mathrm{pa}}$.

THEOREM 6.6 (Full Abstraction). *Let P, Q be programs. Then*

$$P \approx_{\mathrm{pa}} Q \quad \text{if} \quad \mathscr{F}_{\mathrm{pa}}(P) = \mathscr{F}_{\mathrm{pa}}(Q).$$

Proof. Let us define the set of all atoms containing only distinct variables as $F = \{q(\tilde{X}) \mid q \in \Pi\}$. In order to prove the thesis we need three preliminary facts.

First, by definition of F, for any P there exists a derivation $G \overset{\vartheta}{\rightsquigarrow}_P G'$ iff there exists a derivation $G \overset{\vartheta}{\rightsquigarrow}_P G' \overset{\varepsilon}{\rightsquigarrow}_P \square$ (where ε is the empty substitution). This implies that ϑ is a partial answer for G in P iff ϑ is a computed answer substitution for G in $P \cup F$. Therefore, from definitions we can derive the following fact

$$P_1 \approx_{\mathrm{pa}} P_2 \quad \text{iff} \quad P_1 \cup F \approx_{\mathrm{ca}} P_2 \cup F. \tag{1}$$

Moreover, the following holds

$$H: -\tilde{L} \in \iota(\mathscr{F}_{\mathrm{pa}}(P \cup F)) \quad \text{iff} \quad H \in \iota(\mathscr{F}_{\mathrm{ca}}(P \cup F)), \tag{2}$$

since

$H: -\tilde{B} \in \iota(\mathscr{F}_{\mathrm{pa}}(P \cup F))$

iff (by definition of $\mathscr{F}_{\mathrm{pa}}$)

there exists $H: -\tilde{C} \in \iota(\mathscr{F}(P \cup F))$

iff (by definition of $\mathscr{F}_{\mathrm{pa}}$ and F)

$H \in \iota(\mathscr{F}_{\mathrm{ca}}(P \cup F))$.

Finally, since in the definition of $\sim_{\mathrm{pa}}$ we use the set Id_{Π} (see Definition 6.2), from the definitions of F and Id_{Π} we can derive

$$\mathscr{F}_{\mathrm{pa}}(P \cup F) = \mathscr{F}_{\mathrm{pa}}(P). \tag{3}$$

In order to prove the thesis we can now use the full abstraction of the s-semantics,

$P \approx_{\mathrm{pa}} Q$

iff (from (1) and Theroem 4.6)

$\mathscr{F}_{\mathrm{ca}}(P \cup F) = \mathscr{F}_{\mathrm{ca}}(Q \cup F)$

iff (from (2))

$\mathscr{F}_{\mathrm{pa}}(P \cup F) = \mathscr{F}_{\mathrm{pa}}(Q \cup F)$

iff (from (3))

$\mathscr{F}_{\mathrm{pa}}(P) = \mathscr{F}_{\mathrm{pa}}(Q)$

and this completes the proof. ∎

For $\mathscr{F}_{\mathrm{cpa}}(P)$ we obtain only the following correctness result. The problems for obtaining full abstraction here are the same as those mentioned in Section 5.

PROPOSITION 6.7. *Let P, Q be programs. If $\mathscr{F}_{\mathrm{cpa}}(P) = \mathscr{F}_{\mathrm{cpa}}(Q)$ then $P \approx_{\mathrm{cpa}} Q$.*

Proof. Straightforward by Lemma 6.5 and by Definition 2.4. ∎

The information needed to model call patterns can be obtained from the clauses in $\mathscr{F}(P)$ as well. For example, if $H: -B_1, \ldots, B_n \in \mathscr{F}(P)$ and $\vartheta = \mathrm{mgu}(A, H)$ then $B_i\vartheta$ is a call pattern for the goal A. Since we are not considering a specific selection rule, we only need the information on the relation between the head and the various atoms in the body. In other words, the clause $H: -B_1, \ldots, B_n$ is equivalent to the set of clauses $\{H: -B_1, \ldots, H: -B_n\}$. In the case of correct call patterns we need to keep all the information on the clause bodies (viewed as multisets). Therefore we need the following.

DEFINITION 6.8. Assume that $c \in \mathscr{C}$ and $c = H: -B_1, \ldots, B_n \in \mathscr{C}$. Then we define

$$\mathrm{Krom}(c) = \bigcup_{i=1,\ldots,n} \{H: -B_i \in \mathscr{C}\} \cup \{H \in \mathscr{A}\}$$

The *Krom* operator, which transforms (equivalence classes of) clauses into sets of binary clauses (called also Krom clauses), is extended in the obvious way to subsets of $\mathscr{C}$.

DEFINITION 6.9. Let P be a program and let $I, J \subseteq \mathscr{C}$. We define $I \sim_{\mathrm{pt}} J$ iff $\mathrm{Krom}(I) = \mathrm{Krom}(J)$. Moreover, we define $I \sim_{\mathrm{cpt}} J$ iff $I = J$.

The usual properties on the $\sim$ equivalences are ensured by the following.

PROPOSITION 6.10. *Let $I, J \subseteq \mathscr{C}$. The following properties hold:*

(i) $\sim_{\mathrm{pt}}$ *is a congruence wrt infinite unions,*

(ii) *if* $I \sim_{\mathrm{pt}} J$ *then* $\Gamma_P(I) \sim_{\mathrm{pt}} \Gamma_P(J)$.

Hence we have the usual definition of the semantics as instances of the scheme.

DEFINITION 6.11 ((Correct) Call Patterns Semantics). Let P be a program. The semantics $\mathscr{F}_{\mathrm{pt}}(P)$ for call patterns and $\mathscr{F}_{\mathrm{cpt}}(P)$ for correct call patterns are defined as the instances of Definition 3.10 obtained by using $\sim_{\mathrm{pt}}$ and $\sim_{\mathrm{cpt}}$, respectively.

The following lemma shows how to extract (correct) call patterns from the semantics.

LEMMA 6.12. *Let P be a program and let G be a goal.*

1. *A is a call pattern for G in P iff there exists a subgoal $G' = A_1, \ldots, A_n$ of G and there exists a sequence $\tilde{H}$ of heads of clauses $H_i: -\tilde{B}_i$ in $\iota(\mathscr{F}_{\mathrm{pt}}(P))$, $i = 1, \ldots, n$, such that $\gamma = \mathrm{mgu}(G', \tilde{H})$ and $A = B\gamma$, where B is an atom in the goal G'' obtained from G by replacing each A_i with $\tilde{B}_i$,*

2. *A is a correct call pattern for G in P iff there exists a subgoal $G' = A_1, \ldots, A_n$ of G and there exists a sequence $\tilde{H}$ of heads of clauses $H_i: -\tilde{B}_i$ in $\iota(\mathscr{F}_{\mathrm{cpt}}(P))$, $i = 1, \ldots, n$, such that*

(a) *$\gamma = \mathrm{mgu}(G', \tilde{H})$ and $A = B\gamma$, where B is an atom in the goal G'' defined as before,*

(b) *there exists a sequence $\mathscr{C}$ of atoms in $\iota(\mathscr{F}_{\mathrm{cpt}}(P))$ such that $G''\gamma$ and $\mathscr{C}$ are unifiable.*

Proof. The proof is similar to that of Lemma 6.5. ∎

Finally, the following proposition shows the correctness results for the call pattern semantics.

PROPOSITION 6.13. *Let P_1, P_2 be programs. Then the following statements hold:*

1. *if $\mathscr{F}_{\mathrm{pt}}(P_1) = \mathscr{F}_{\mathrm{pt}}(P_2)$ then $P_1 \approx_{\mathrm{pt}} P_2$,*
2. *if $\mathscr{F}_{\mathrm{cpt}}(P_1) = \mathscr{F}_{\mathrm{cpt}}(P_2)$ then $P_1 \approx_{\mathrm{cpt}} P_2$.*

Proof. The proof is straightforward from Lemma 6.12. ∎

7. CONCLUSIONS

In this paper we have introduced a general semantic scheme $\mathscr{F}(P)$ which can equivalently be characterized by an unfolding construction and as the least fixpoint of a suitable operator. The scheme is defined on a semantic domain consisting of $\sim$ equivalences classes of clauses and is parametric wrt the equivalence $\sim$. Correct and fully abstract compositional semantics for specific observables can be obtained from the scheme by using suitable $\sim$ equivalences.

We have then considered several observational equivalences and investigated how they are related. In particular, for each observational equivalence we have shown a correct (in some cases fully abstract) semantics obtained as an instance of the scheme.

The relations among the various equivalences we have studied and the correctness and full abstraction results are summarized in Fig. 3. In the figure operators and semantics denote the induced equivalences on programs. For example, T^s_P denotes the (extensional) equivalence $T^s_P = T^s_Q$, while $\mathscr{F}^s_P$ denotes the equivalence $\mathscr{F}^s_P = \mathscr{F}^s_Q$. Arrows denote strict inclusions of the equivalences, i.e., $A \to B$ means that A is strictly finer than B. Equivalences included in a box coincide. Columns correspond to increasingly more concrete observables (from the left to the right). Namely, we have successful derivations (on the leftmost column), computed answers, partial answers, and call patterns (on the rightmost column). Each column contains, from the top to the bottom, increasingly coarser equivalences and more abstract semantics for the same observable.

Fully abstract semantics wrt $\approx_X$ are those for which $\mathscr{F}_X(P)$ is in the same box as $\approx_X$, while $\mathscr{F}_Y(P)$ is correct wrt $\approx_X$ if $\mathscr{F}_Y(P) \xrightarrow{*} \approx_X$ holds (where $\xrightarrow{*}$ is the transitive closure of the relation $\to$). For example, the semantics $\mathscr{F}_{\mathrm{pa}}(P)$ is fully abstract wrt (the equivalence induced by) partial answers, while $\mathscr{F}_{(H,\mathrm{ca})}(P)$ is correct wrt computed answers and is compositional wrt union of programs. As shown in the figure, $\mathscr{F}_{(H,\mathrm{ca})}(P)$, $\mathscr{F}_{\mathrm{cpa}}(P)$, $\mathscr{F}_{\mathrm{pt}}(P)$, and $\mathscr{F}_{\mathrm{cpt}}(P)$ are only correct, while all the other semantics obtained as instances of the scheme are fully abstract.

Note that, in the first two columns on the left, all the semantics above the bottom row are compositional wrt program union. For instance, if we consider T^{ca}_P as the semantic of P, such a semantics is correct wrt $\approx_{(H,\mathrm{ca})}$, i.e., it correctly models computed answers and it is compositional wrt $\bigcup_H$.

We have shown that the semantic scheme $\mathscr{F}(P)$ allows us to model several observables. $\mathscr{F}(P)$ could then be taken as the basic semantic framework for studying program transformation and program analysis. In fact, as previously mentioned, even if here we restrict ourselves to "concrete" observables, we argue that similar constructions can be done for "abstract observables" such as those arising in program analysis. *Abstract interpretation* could be understood in terms of the choice of a suitable observable and therefore of a suitable $\sim$ equivalence. Thus the scheme could be used to uniformly represent and to compare both "abstract" and "concrete" semantics. Since $\mathscr{F}(P)$ has both a procedural and a fixpoint definition, equivalent top-down and bottom-up techniques are available.

Finally, we mention that, as discussed in Section 2, observables such as partial answers and call patterns have also a natural definition which is dependent on the selection rule. The adaptation of our framework to cope with a specific selection rule is not difficult and has been studied in [22]. An interesting open problem is the definition of a unique scheme treating both cases in a uniform way.

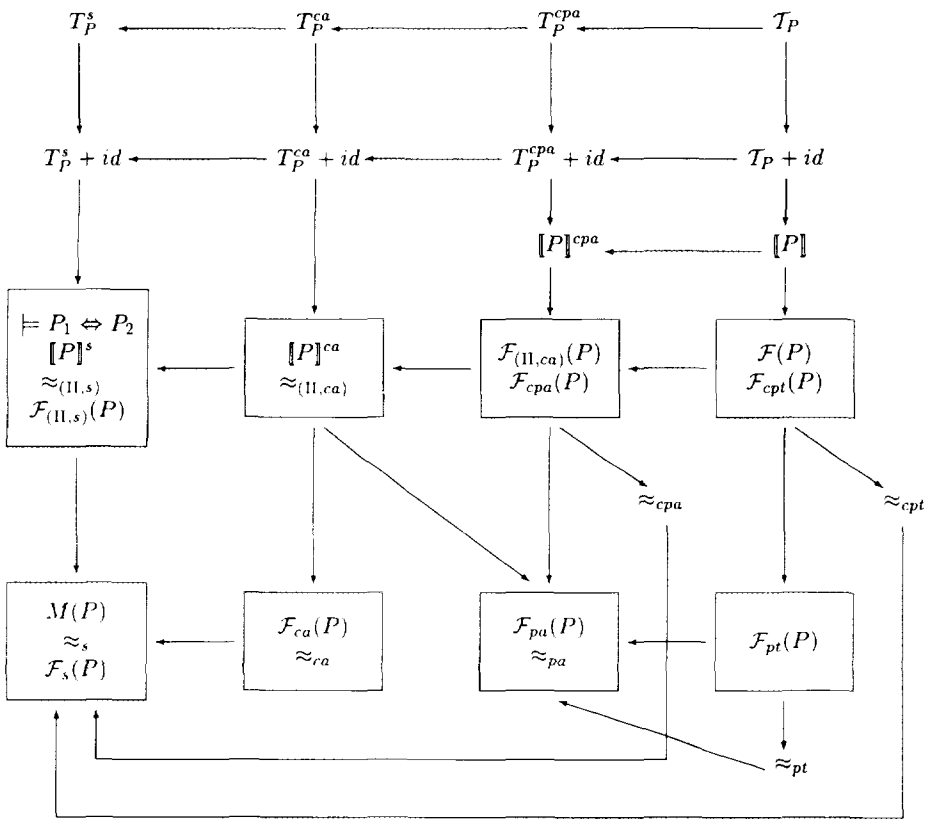

FIG. 3. Relations among equivalences and semantics. Each operator on semantics denotes the induced equivalence on programs. $A \to B$ means that A is strictly finer than B. Equivalences included in a box coincide. The leftmost column data are from [30].

A. APPENDIX

In this appendix we give the proofs which were missing in the previous sections. We start by proving the stated relations among observational equivalences.

PROPOSITION 2.6. *Let P and Q be programs.*

(i) *If $P \approx_{\text{cpa}} Q$ then $P \approx_s Q$.*

(ii) *If $P \approx_{\text{cpt}} Q$ then $P \approx_s Q$.*

(iii) *If $P \approx_{(\Pi,\text{ca})} Q$ then $P \approx_{\text{pa}} Q$.*

(iv) *If P and Q are finite programs defined on a signature which contains infinitely many unary function symbols, $P \approx_{\text{pt}} Q$ implies $P \approx_{\text{pa}} Q$.*

Proof. (i) We prove the contrapositive. Assume $P \not\approx_s Q$. Then there exists a goal G which has a refutation $G \overset{\vartheta}{\mapsto}_P \square$ and which has no refutation in Q. By soundness of SLD resolution, ϑ is a correct answer for G in P. Therefore by [28] there exists a substitution β such that $\vartheta\beta$ is a correct answer for G and $G\vartheta\beta = G'$ is ground. By completeness of SLD resolution and since G' is ground, there exists a refutation $G' \overset{\varepsilon}{\mapsto}_P \square$. Since G' is an instance of G, the hypothesis on Q and the lifting lemma [28] imply that G' has no refutation in Q. Then ε is a correct partial answer for G' in P, while ε is not a correct partial answer for G' in Q. This means that $P \not\approx_{\text{cpa}} Q$ and completes the proof.

(ii) If the goal G has a refutation in P then any atom in G is a correct call pattern in P. Since P and Q have the same correct call patterns, G has also a refutation in Q. Then the thesis follows by symmetry.

(iii) We prove the contrapositive. Assume $P \not\approx_{\text{pa}} Q$. Then, by Theorem 6.6, $\mathcal{F}_{\text{pa}}(P) \neq \mathcal{F}_{\text{pa}}(Q)$, i.e. there exists a clause H: $-\tilde{B} \in \iota(\mathcal{F}_{\text{pa}}(P)) \cup \text{Id}_\Pi$ such that there is no clause K: $-\tilde{C} \in \iota(\mathcal{F}_{\text{pa}}(Q)) \cup \text{Id}_\Pi$ with $H = K\rho$ for a renaming ρ. Assume that $p = \text{Pred}(H)$. By definition of Id_Π, H is not a renamed version of $p(\tilde{X})$ and therefore $\vartheta = \text{mgu}(H, p(\tilde{X}))$ implies $\vartheta \neq \varepsilon$. Let us consider now the set of all the atoms containing only distinct variables

$$F = \{q(\tilde{Y}) \mid \tilde{Y} \text{ are distinct variables and } q \in \Pi\}.$$

It is easy to check that $H \in \iota(\mathcal{F}_{\text{ca}}(P \cup F))$ and $H \notin \iota(\mathcal{F}_{\text{ca}}(Q \cup F))$. By the strong completeness theorem in [17], $P \cup F \not\approx_{\text{ca}} Q \cup F$ and therefore, by definition of $\approx_{(\Pi,\text{ca})}$, $P \not\approx_{(\Pi,\text{ca})} Q$. This completes the proof.

(iv) We prove the contrapositive. Assume $P \not\approx_{\text{pa}} Q$ and let H: $-\tilde{B} \in \iota(\mathcal{F}_{\text{pa}}(P)) \cup \text{Id}_\Pi$, $p = \text{Pred}(H)$ and $\vartheta = \text{mgu}(H, p(\tilde{X}))$, $\vartheta \neq \varepsilon$, be defined as in (iii). By Lemma 6.5,

ϑ is a partial answer for $p(\tilde{X})$ in P, while ϑ is not a partial answer for $p(\tilde{X})$ in Q. Let us consider the goal $G = p(\tilde{X}), p(f(X_1), ..., f(X_n))$, where f is a new unary function symbol. To prove the thesis it is sufficient to observe now that $p(f(X_1), ..., f(X_n))\,\vartheta$ is a call pattern for G in a program R iff ϑ is a partial answer for $p(\tilde{X})$ in R. In fact, since f is a new function symbol, the variables $X_1, ..., X_n$ cannot be instantiated by the evaluation of $p(f(X_1), ..., f(X_n))$. This completes the proof. ∎

We prove now that the set of interpretations is a complete lattice and the continuity of the $\mathscr{T}_P$ operator.

LEMMA 3.5. *The relation* $\sqsubseteq$ *is an ordering on* $\mathfrak{I}$. *Moreover,* $(\mathfrak{I}, \sqsubseteq)$ *is a complete lattice (with* $\sqcup$ *as lub and* $[\varnothing]$ *as bottom).*

Proof. (*Reflexivity*) The proof is straightforward.

(*Transitivity*) Let us assume that for $I, J, Z \in \mathfrak{I}$, $I \sqsubseteq J$ and $J \sqsubseteq Z$. By Definition 3.4, for $I, J \in \mathfrak{I}$, $I \sqsubseteq J$ iff $\iota(I) \cup \iota(J) \sim \iota(J)$ (for any $\iota(I), \iota(J)$). Since $\sim$ is a congruence wrt $\cup$ we have

$$\begin{aligned} &\iota(I) \cup \iota(Z) \\ &\quad \sim \iota(I) \cup \iota(J) \cup \iota(Z) && \text{(since } J \sqsubseteq Z\text{)} \\ &\quad \sim \iota(J) \cup \iota(Z) && \text{(since } I \sqsubseteq J\text{)} \\ &\quad \sim \iota(Z) && \text{(since } J \sqsubseteq Z\text{)} \end{aligned}$$

and therefore $I \sqsubseteq Z$.

(*Antisymmetricity*) If $I \sqsubseteq J$ and $J \sqsubseteq I$ then $\iota(I) \sim \iota(I) \cup \iota(J) \sim \iota(J)$ and therefore $I = J$.

Let us now prove that for any $X \subseteq \mathfrak{I}$ (including the empty set) $\bigsqcup X = \mathrm{lub}(X)$. This is all we need to prove that $\mathfrak{I}$ is a complete lattice since we can obtain the greatest lower bound as $\mathrm{glb}(X) = \mathrm{lub}\{I \in \mathfrak{I} \mid \forall J \in X, I \sqsubseteq J\}$. Let $X \subseteq \mathfrak{I}$ and let $L = \bigsqcup X$. For $J \in X$ we have the following equalities

$$\begin{aligned} J \sqcup L &= [\iota(J) \cup \iota(L)] && \text{(by definition of } \sqcup\text{)} \\ &= \left[\iota(J) \cup \iota\left(\left[\bigcup_{I \in X} \iota(I)\right]\right)\right] \\ &\quad \text{(by definition of } L\text{)} \\ &= \left[\iota(J) \cup \left(\bigcup_{I \in X} \iota(X)\right)\right] \\ &\quad \text{(by definition of } \iota \text{ and } \sim\text{)} \\ &= \left[\bigcup_{I \in X} \iota(I)\right] && \text{(since } J \in X\text{)} \\ &= L && \text{(by definition of } L\text{).} \end{aligned}$$

Therefore, by Definition 3.4, if $J \in X$ then $J \sqsubseteq \bigsqcup X$. Let us assume now that U is an upper bound of X, i.e., $\forall I \in X$, $I \sqsubseteq U$. By the definition of $\sqsubseteq$ (for any $\iota(I), \iota(U)$),

$$\forall I \in X, \qquad \iota(I) \cup \iota(U) \sim \iota(U). \tag{4}$$

Then we have the following equalities for $L = \bigsqcup X$:

$$\begin{aligned} U \sqcup L &= [\iota(U) \cup \iota(L)] && \text{(by definition of } \sqcup\text{)} \\ &= \left[\iota(U) \cup \iota\left(\left[\bigcup_{I \in X} \iota(I)\right]\right)\right] && \text{(by definition of } L\text{)} \\ &= \left[\iota(U) \cup \left(\bigcup_{I \in X} \iota(I)\right)\right] \\ &\quad \text{(by definition of } \iota \text{ and } \sim\text{)} \\ &= [\iota(U)] && \text{(by (1))} \\ &= U && \text{(by definition of } \iota \text{ and } \sim\text{)} \end{aligned}$$

Hence, by Definition 3.4, if U is an upper bound of X then $\sqcup X \sqsubseteq U$. Then $\bigsqcup X = \mathrm{lub}(X)$ and this completes the proof. ∎

LEMMA 3.9. *Let* P *be a program. Then* $\mathscr{T}_P$ is continuous on $(\mathfrak{I}, \sqsubseteq)$ *and* $\mathscr{T}_P \uparrow \omega$ *is the least fixpoint of* $\mathscr{T}_P$.

Proof. We prove the continuity. The second part follows from well-known results of lattice theory. Let X be a directed subset of $\mathfrak{I}$. We have to prove that $\mathscr{T}_P(\bigsqcup X) = \bigsqcup\{\mathscr{T}_P(I) \mid I \in X\}$, i.e., that $\iota(\mathscr{T}_P(\bigsqcup X)) \sim \iota(\bigsqcup\{\mathscr{T}_P(I) \mid I \in X\})$. We have the following implications, where X_f denotes a finite subset of X.

$$\begin{aligned} &A \in \iota(\mathscr{T}_P(\bigsqcup X)) \\ &\quad \text{iff (by Definition 3.8)} \\ &A \in \iota([\Gamma_P(\iota(\bigsqcup X))]) \\ &\quad \text{iff (by definition of } \bigsqcup X\text{)} \\ &A \in \iota\left(\left[\Gamma_P\left(\iota\left(\left[\bigcup_{I \in X} \iota(I)\right]\right)\right)\right]\right) \\ &\quad \text{iff (since } \sim \text{ is a congruence wrt } \Gamma_P\text{)} \\ &A \in \iota\left(\left[\Gamma_P\left(\bigcup_{I \in X} \iota(I)\right)\right]\right) \\ &\quad \text{iff (by definition of } \Gamma_P\text{,} \\ &\quad \text{since clauses have finite bodies)} \\ &A \in \iota\left(\left[\Gamma_P\left(\bigcup_{I \in X_f} \iota(I)\right)\right]\right) \\ &\quad \text{iff (since } \sim \text{ is a congruence wrt } \Gamma_P\text{)} \end{aligned}$$

$$A \in \iota\left(\left[\Gamma_P \iota\left(\left[\bigcup_{I \in X_f} \iota(I)\right]\right)\right]\right)$$

iff (by definition of $\bigsqcup$)

$A \in \iota([\Gamma_P \iota(\bigsqcup X_f)])$

iff (since X is directed)

$\exists J \in X$ s.t. $A \in \iota([\Gamma_P(J)])$

iff (by Definition 3.8)

$\exists J \in X$ s.t. $A \in \iota(\mathcal{T}_P(J))$

iff (by definition of $\cup$)

$$A \in \bigcup_{I \in X} \iota(\mathcal{T}_P(I)).$$

Then

$$\bigcup_{I \in X} \iota(\mathcal{T}_P(I)) = \iota(\mathcal{T}_P(\bigsqcup X)). \tag{5}$$

Note that for a set of clauses C, $C \sim \iota([C])$. Therefore

$$\begin{aligned}
&\iota(\bigsqcup \{\mathcal{T}_P(I) \mid I \in X\}) \\
&\quad = \iota\left(\left[\bigcup_{I \in X} \iota(\mathcal{T}_P(I))\right]\right) && \text{(by definition of } \bigsqcup) \\
&\quad \sim \bigcup_{I \in X} \iota(\mathcal{T}_P(I)) && \text{(by the previous remark)} \\
&\quad = \iota(\mathcal{T}_P(\bigsqcup X)) && \text{(by (5))}
\end{aligned}$$

and this concludes the proof. ∎

In order to prove Proposition 3.12 we now need some definitions and results about substitutions and equations from [15, 27, 34]. An equation is an atom $s = t$, where s, t are terms and $=$ is a predicate symbol which is interpreted as the syntactic equality on the Herbrand universe. If $\tilde{s}$ denotes the sequence of terms $s_1, ..., s_n$ and $\tilde{t}$ denotes $t_1, ..., t_n$ then $\tilde{t} = \tilde{s}$ denotes the equations $s_1 = t_1, ..., s_n = t_n$. Given a set of equations $E = \{s_1 = t_1, ..., s_n = t_n\}$, the (most general) unifier of E is a (most general) unifier of $((s_1, ..., s_n), (t_1, ..., t_n))$. It is well known that the mgu of a set of equations (terms) is unique up to renaming (see for example [15]). Moreover, if a set of equations E is unifiable, then there exists an idempotent mgu of E. A solution of E is a grounding unifier of E (i.e., a unifier ϑ such that $E\vartheta$ contains no variable). Two sets of equations E_1, E_2 are equivalent ($E_1 \approx E_2$) if they have the same solutions. The lattice structure on idempotent substitutions [15] is isomorphic to the lattice structure on equations introduced in [27]. Therefore we can indifferently use equations or idempotent mgus.

DEFINITION A.1. Let $\vartheta = \{X_1/t_1, ..., X_n/t_n\}$ be a substitution. Then we define $\mathcal{E}(\vartheta) = \{X_1 = t_1, ..., X_n = t_n\}$.

LEMMA A.2 [34]. *Let ϑ_1, ϑ_2 be idempotent substitutions. Then* $\mathrm{mgu}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) = \vartheta_1\, \mathrm{mgu}(\mathcal{E}(\vartheta_2)\vartheta_1) = \vartheta_2\, \mathrm{mgu}(\mathcal{E}(\vartheta_1)\vartheta_2)$.

THEOREM A.3 [27]. *If the Herbrand universe is nontrivial, then ϑ is an idempotent* mgu *of E iff $\mathcal{E}(\vartheta) \approx E$.*

From the above results we can easily derive the following corollaries.

COROLLARY A.4 [8]. *If the Herbrand universe is nontrivial and if $E_1 \approx E_2$, then for any set of equations E,* $\mathrm{mgu}(E \cup E_1) = \mathrm{mgu}(E \cup E_2)$.

COROLLARY A.5 [8]. *Let E_1, E_2 be sets of equations with* $\mathrm{mgu}(E_1) = \vartheta$ *and* $\mathrm{mgu}(E_2\vartheta) = \gamma$. *Then* $\mathrm{mgu}(E_1 \cup E_2) = \vartheta\gamma$.

In the following we will always implicitly consider idempotent mgus and nontrivial Herbrand universe.

In order to prove Proposition 3.12 we will consider also an equational version [37] of SLD derivation, denoted by $\rightsquigarrow^*_{P,R}$, which uses equations instead of mgus. The previous stated isomorphism allows us to prove the equivalence of the two versions of SLD (see below). More precisely, given a goal $p(\tilde{s})$, G, E, where G is a conjunction of atoms, E is a set of equations, and $p(\tilde{s})$ is the selected atom, and given a clause $p(\tilde{t})$: $-B_1, ..., B_n$, we can define $p(\tilde{s})$, G, $E \rightsquigarrow^*_{P,R} B_1, ..., B_n$, G, $\tilde{s} = \tilde{t}$, E in one step iff the set $E \cup \{\tilde{s} = \tilde{t}\}$ is unifiable. Provided that no equational atom is chosen by the selection rule R and by replacing the atom $p(\tilde{t})$ by $p(\tilde{X})$, $\tilde{X} = \tilde{t}$ (where $\tilde{X}$ are new variables), we can then replace a derivation $p(\tilde{t}) \overset{\vartheta}{\rightsquigarrow}_{P,R} G'$ by the equivalent derivation $p(\tilde{X})$, $\tilde{X} = \tilde{t} \rightsquigarrow^*_{P,R} G$, E which uses clauses the same as those of the previous one. The equivalence is formally stated by the following.

LEMMA A.6 [37]. *Let P be a program and $G = p_1(\tilde{t}_1), ..., p_k(\tilde{t}_k)$ be a goal. Assume $H = p_1(\tilde{X}_1), ..., p_k(\tilde{X}_k)$, $E = \{\tilde{X}_1 = \tilde{t}_1, ..., \tilde{X}_k = \tilde{t}_k\}$, and let $\beta = \{x/t \mid x = t \in E\}$, with $\tilde{X}_1, ..., \tilde{X}_k$ new distinct variables. Then there exists a derivation $G \overset{\vartheta}{\rightsquigarrow}_{P,R} G'$ iff there exists a derivation H, $E \rightsquigarrow^*_{P,R} H'$, E', where $\beta\vartheta = \mathrm{mgu}(E')$ and $G' = H'\beta\vartheta$. Moreover,* $\mathrm{mgu}(E')_{|\mathrm{Var}(G)} = \vartheta_{|\mathrm{Var}(G)}$.

Note that the previous lemma holds for any selection rule. Moreover, since in the $\rightsquigarrow^*_{P,R}$ derivations the computation is performed by accumulating equations, clearly the order in which equations are added is not relevant. Therefore we have the following.

PROPOSITION 3.12. *Let P be a program and G be a goal. Assume that the semantic domain is defined by letting $\sim$ be the identity. Then there exist R, ϑ and a derivation $G \overset{\vartheta}{\rightsquigarrow}_{P,R} \tilde{B}$ with $\vartheta_{|\mathrm{Var}(G)} = \gamma$ iff there exist a non-empty subgoal*

$G' = A_1, ..., A_n$ of G and a sequence $\tilde{H}$ of heads of n clauses $H_j: -\tilde{D}_j$ in $\iota(\mathscr{F}(P))$ such that

1. $\xi = \text{mgu}(G', \tilde{H})$ *and* $\gamma = \xi_{|\text{Var}(G)}$,
2. $\{|G''\xi|\} = \{|B|\}$ *where G'' denotes the goal obtained from G by replacing each A_j by $\tilde{D}_j$.*

Proof. In the following we will use the following notation. G is the goal $p_1(\tilde{t}_1), ..., p_k(\tilde{t}_k)$ and G_x denotes the goal $p_1(\tilde{X}_1), ..., p_k(\tilde{X}_k)$ where $\tilde{X}_i$, for $1 \leqslant i \leqslant k$ are new distinct variables. Moreover, for i as before, E_i is the set of equations $\{\tilde{X}_i = \tilde{t}_i\}$, $\beta_i = \{x/t \mid x = t \in E_i\}$, and $E = \bigcup_{i=1,...,k}(E_i)$. By Lemma A.6 we have that

(a) there exists a derivation $G \overset{\vartheta}{\rightsquigarrow}_{P,R} \tilde{B}$ iff there exists a derivation G_x, $E \rightsquigarrow^*_{P,R} \tilde{B}_x$, $E \cup E^*$ where, for $\gamma = \text{mgu}(E \cup E^*)$, $\tilde{B} = \tilde{B}_x\gamma$ and $\vartheta_{|\text{Var}(G)} = \gamma_{|\text{Var}(G)}$.

Since in the $\rightsquigarrow^*_{P,R}$ derivation the computation is performed by accumulating sets of equations, clearly the final resolvent of a derivation D is independent from the order in which atoms are rewritten in D. Therefore a straightforward inductive argument gives that

(b) there exists a derivation G_x, $E \rightsquigarrow^*_{P,R} \tilde{B}_x$, $E \cup E^*$ iff there exists a non-empty set of n indexes $J = \{j_1, ..., j_n\} \subseteq \{1, ..., k\}$, such that the following conditions hold:

1. for any $j \in J$ there exists R_j such that there exists a derivation $p_j(\tilde{X}_j)$, $E_j \rightsquigarrow^*_{P,R_j} \tilde{L}_j$, $E_j \cup E_j^*$,
2. $\bigcup_{j \in J}(E_j^*) = E^*$ and $\{|G''_x|\} = \{|\tilde{B}_x|\}$,

where G''_x is obtained from G_x by replacing $p_j(\tilde{X}_j)$ by $\tilde{L}_j$ for any $j \in J$.

Let us denote by K the set of indexes $\{1, ..., k\}$. Assume $\delta_j = \text{mgu}(E_j^*)$ for $j \in J$ and $\delta_i = \varepsilon$ for $i \in K \backslash J$. By the definition of $\mathscr{F}(P)$ and by Lemma A.6, for any $j \in J$ the following relation holds:

$$\begin{aligned}&\text{there exists a derivation } p_j(\tilde{X}_j) \rightsquigarrow^*_{P,R_j} \tilde{L}_j,\ E_j^* \\ &\quad \text{iff there exists a clause } c_j: p_j(\tilde{X}_j)\,\delta_j: -\tilde{L}_j\delta_j \in \mathscr{F}(P) \qquad (1)\end{aligned}$$

(note that $\tilde{L}_j\delta_j$ is $\tilde{D}_j$ as defined in the hypothesis of the proposition). Let us denote by W the set $\text{Var}(G)$. We have

$$\begin{aligned}&\gamma && (2)\\ &= \text{mgu}(E \cup E^*) \quad \text{(by definition)} && (3)\\ &= \text{mgu}\left(\bigcup_{j \in K}(\{\tilde{t}_j = \tilde{X}_j\} \cup \mathscr{E}(\delta_j))\right) \\ &\quad \text{(by definition of } E^*\text{, Theorem A.3,} \\ &\quad \text{and Corollary A.4)} && (4)\end{aligned}$$

Since $\tilde{X}_j$ are new variables by hypothesis, by definition of δ_j, for any $i, j \in K$, $\text{Var}(\tilde{t}_i) \cap \text{Var}(\delta_j) = \varnothing$ and $i \neq j$ implies $\text{Var}(\tilde{X}_i) \cap \text{Var}(\delta_j) = \varnothing$. Moreover, for any $j \in K$, $\tilde{X}_j \cap W = \varnothing$ and $\text{Var}(\delta_j) \cap W = \varnothing$. Then for any $j \in J$,

$$\begin{aligned}&\text{mgu}\left(\bigcup_{i \in K}\{\tilde{t}_i = \tilde{X}_i\} \cup \mathscr{E}(\delta_j)\right) \\ &\quad = \delta_j\,\text{mgu}\left(\bigcup_{i \in K}\{\tilde{t}_i\delta_j = \tilde{X}_i\delta_j\}\right) \quad \text{(by Corollary A.5)} \\ &\quad = \delta_j\,\text{mgu}(\{\tilde{t}_1 = \tilde{X}_1, ..., \tilde{t}_j = \tilde{X}_j\delta_j, ..., \tilde{t}_k = \tilde{X}_k\}) \\ &\quad \text{(by the previous remarks on the variables)} \qquad (5)\end{aligned}$$

Since $\text{Var}(\delta_j) \cap \text{Var}(\delta_i) = \varnothing$ for $i \neq j \in K$, by repeated applications of the previous equality we have

$$\begin{aligned}&\text{mgu}\left(\bigcup_{j \in K}\{\tilde{t}_j = \tilde{X}_j\} \cup \mathscr{E}(\delta_j)\right) \\ &\quad = \delta_1 \cdots \delta_k\,\text{mgu}\left(\bigcup_{j \in K}\{\tilde{t}_j = \tilde{X}_j\delta_j\}\right). \qquad (6)\end{aligned}$$

By denoting $\text{mgu}(\bigcup_{j \in K}\{\tilde{t}_j = \tilde{X}_j\delta_j\})$ by α, from (2) and (6) we derive

$$\delta_1 \cdots \delta_k\alpha = \gamma. \qquad (7)$$

The hypothesis of the existence of γ, the equality (7), and the definition of α imply that there exists

$$\xi = \text{mgu}(G', \tilde{H}),$$

where G' is the subgoal of G obtained by taking the atoms A_j for $j \in J$ and $\tilde{H}$ is the sequence of the heads of the clauses $c_j \in \mathscr{F}(P)$, as defined in (1). Moreover,

$$\begin{aligned}&\xi_{|W} = \text{mgu}\left(\bigcup_{j \in J}\{\tilde{t}_j = \tilde{X}_j\delta_j\}\right)_{|W} \\ &\quad \text{(by definition)} \\ &\quad = \text{mgu}\left(\bigcup_{j \in K}\{\tilde{t}_j = \tilde{X}_j\delta_j\}\right)_{|W} \\ &\quad \text{(by definition of } \delta_i\text{, since } W \cap \tilde{X}_i = \varnothing \text{ for } i \in K \backslash J) \\ &\quad = (\delta_1 \cdots \delta_k\alpha)_{|W} \\ &\quad \text{(since } \text{Var}(\delta_i) \cap W = \varnothing \text{ and by definition of } \alpha) \\ &\quad = \gamma_{|W} \quad \text{(by (7))} \\ &\quad = \vartheta_{|W} \quad \text{(by (a) above)} \qquad (8)\end{aligned}$$

and this completes the proof of the first part of the thesis.

We have now to prove that $\{|G''\xi|\} = \{|\tilde{B}|\}$ where where G'' denotes the goal obtained from G by replacing each A_j by $\tilde{L}_j\delta_j$, for any $j \in J$.

Note that, by definition of G''_χ, G'', and δ_i, B_i is an atom in G'' iff, either $B_i = p_i(\tilde{t}_i)$, for $i \in K \backslash J$ and $p_i(\tilde{X}_i)$ is in G''_χ, or $B_i \in \tilde{L}_j \delta_j$, for $j \in J$ and $\tilde{L}_j$ is in G''_χ. Therefore, by the third equality in (b) and by (a) above, we only need to show that for any $i \in K \backslash J$, $p_i(\tilde{t}_i)\,\xi = p_i(\tilde{X}_i)\,\gamma$ and, for any $j \in J$, $\tilde{L}_j \delta_j \xi = \tilde{L}_j \gamma$. Let us consider the two cases separately.

(i) First observe that, by Corollary A.5 and by definition of β_i,

$$\beta_i \tau = \gamma, \tag{9}$$

where $\tau = \mathrm{mgu}(E \cup E^* \backslash \{\tilde{X}_i = \tilde{t}_i\})$ $\beta_i = \mathrm{mgu}(E \cup E^* \backslash \{\tilde{X}_i = \tilde{t}_i\})$. Moreover, by definition of β_i,

$$\tilde{t}_i \gamma = \tilde{t}_i \tau. \tag{10}$$

By (9), the definition of β_i, and (10) we derive

$$p_i(\tilde{X}_i)\,\gamma = p_i(\tilde{t}_i)\,\gamma. \tag{11}$$

Then, for $i \in K \backslash J$, we have

$$\begin{aligned} p_i(\tilde{t}_i)\,\xi &= p_i(\tilde{t}_i)\,\xi_{|W} && (\text{since } \mathrm{Var}(p_i(\tilde{t}_i)) \subseteq W) \\ &= p_i(\tilde{t}_i)\,\gamma_{|W} && (\text{by (8)}) \\ &= p_i(\tilde{t}_i)\,\gamma && (\text{since } \mathrm{Var}(p_i(\tilde{t}_i)) \subseteq W) \\ &= p_i(\tilde{X}_i)\,\gamma && (\text{by (11)}). \end{aligned}$$

(ii) Assume now that $j \in J$. Note that, by definition of δ_j, for $h \in J$ and $j \neq h$, $\mathrm{Var}(\tilde{L}_j) \cap \mathrm{Var}(\delta_h) = \varnothing$ and $\mathrm{Var}(\tilde{L}_j \delta_j) \cap \mathrm{Var}(\delta_h) = \varnothing$.

Moreover, note that α differs from ξ only because α can bind some variables $\tilde{X}_i$ with $i \in K \backslash J$. Therefore

$$\tilde{L}_j \delta_j \xi = \tilde{L}_j \delta_j \alpha. \tag{12}$$

We have then the equalities

$$\begin{aligned} \tilde{L}_j \delta_j \xi &= \tilde{L}_j \delta_j \alpha && (\text{by (12)}) \\ &= \tilde{L}_j \delta_1 \cdots \delta_k \alpha \\ &\quad (\text{by previous observations on } \mathrm{Var}(\tilde{L}) \text{ and } \mathrm{Var}(\delta_h)) \\ &= \tilde{L}_j \gamma && (\text{by (7)}) \end{aligned}$$

and this concludes the proof. ∎

Finally, in the following we give the remaining missing proofs.

LEMMA A.7. *Let $\tilde{B}$, $\tilde{D}$ be sequences of atoms, σ be a substitution such that $\tilde{B}\sigma = \tilde{B}$, and $\vartheta = \mathrm{mgu}(\tilde{B}, \tilde{D}\sigma)$. Then the following facts hold:*

1. *$\exists \gamma = \mathrm{mgu}(\tilde{B}, \tilde{D})$ and there exists a substitution α such that $\sigma\vartheta = \gamma\alpha$;*
2. *if σ is a renaming, then α is a renaming.*

Proof. First note that $\tilde{B}$ and $\tilde{D}$ are unifiable since

$$\begin{aligned} \tilde{B}\sigma\vartheta &= \tilde{B}\vartheta && (\text{since } \tilde{B}\sigma = \tilde{B}) \\ &= \tilde{D}\sigma\vartheta && (\text{by definition of } \vartheta). \end{aligned}$$

Then there exists $\gamma = \mathrm{mgu}(\tilde{B}, \tilde{D}) \leqslant \sigma\vartheta$ and the first part of the thesis holds. For the second part observe that, if σ is a renaming, then by definition there exists σ^{-1} such that $\sigma\sigma^{-1} = \varepsilon$. Then the hypothesis $\tilde{B}\sigma = \tilde{B}$ implies $\tilde{B}\sigma^{-1} = \tilde{B}$. Since $\tilde{D}\sigma\sigma^{-1} = \tilde{D}$, the previous equality and the hypothesis on γ imply that $\sigma^{-1}\gamma$ is a unifier of $\tilde{B}$ and $\tilde{D}\sigma$. Hence $\vartheta \leqslant \sigma^{-1}\gamma$, since ϑ is most general. Moreover, $\gamma \leqslant \sigma\vartheta$, by the first part of the thesis. Then we have $\sigma\vartheta \leqslant \sigma\sigma^{-1}\gamma = \gamma \leqslant \sigma\vartheta$ and this implies that there exists a renaming α such that $\gamma\alpha = \sigma\vartheta$ and completes the proof. ∎

PROPOSITION 5.9. *Let $I, J \subseteq \mathscr{C}$. The following properties hold:*

(i) *$\sim_{(\Omega, \mathrm{ca})}$ is a congruence wrt infinite unions,*

(ii) *if $I \sim_{(\Omega, \mathrm{ca})} J$ then $\Gamma_P(I) \sim_{(\Omega, \mathrm{ca})} \Gamma_P(J)$.*

Proof. To prove (i) it is sufficient to observe that, given a set of integers N, if $I_n \sim_{(\Omega, \mathrm{ca})} J_n$ holds for any $n \in N$ and $I_n, J_n \subseteq \mathscr{C}$, then

$$\begin{aligned} \left(\bigcup_{n \in N} I_n\right) \cap \mathscr{C}^\Omega &= \bigcup_{n \in N} (I_n \cap \mathscr{C}^\Omega) && (\text{by set theory}) \\ &= \bigcup_{n \in N} (J_n \cap \mathscr{C}^\Omega) && (\text{by hypothesis}) \\ &= \left(\bigcup_{n \in N} J_n\right) \cap \mathscr{C}^\Omega && (\text{by set theory}). \end{aligned}$$

To prove (ii) we have to show that if c is a clause in $\Gamma_P(I) \cap \mathscr{C}^\Omega$ then there exists a clause c' in $\Gamma_P(J) \cap \mathscr{C}^\Omega$ such that $c' \leqslant_c c$. Let us define $S_I = (I \cap \mathscr{C}^\Omega) \cup \mathrm{Id}_\Omega$ and $S_J = (J \cap \mathscr{C}^\Omega) \cup \mathrm{Id}_\Omega$ and let us assume that $c \in \Gamma_P(I) \cap \mathscr{C}^\Omega$. By definition of Γ_P, there exists a clause $A\colon -\tilde{B} \in P$ denoted by $\bar{c}$ and there exists a sequence $\tilde{H}$ of heads of renamed apart clauses $c_i = H_i\colon -\tilde{L}_i$ in S_I such that

$$\text{there exists } \vartheta = \mathrm{mgu}(\tilde{B}, \tilde{H}) \quad \text{and} \quad c = A\vartheta\colon -\tilde{L}\vartheta, \tag{1}$$

where $\tilde{L}$ denotes the conjunction obtained by replacing each atom B_i in $\tilde{B}$ by $\tilde{L}_i$. Since by hypothesis $I \sim_{(\Omega, \mathrm{ca})} J$, we have that $S_I \simeq S_J$. Then for any i there exists a renamed apart clause $c'_i = H'_i\colon -\tilde{L}'_i$ in S_J such that $c'_i \leqslant_c c_i$.

Let ρ_i be a renaming as in Definition 5.2 for any i. Since the clauses are renamed apart, there exists a renaming ρ

such that $\rho_{|\mathrm{Var}(c_i')} = \rho_{i|\mathrm{Var}(c_i')}$ for any i and $\mathrm{Dom}(\rho) \cap \mathrm{Var}(\bar{c}) = \varnothing$. Therefore, by Lemma A.7 and by (1),

$$\text{there exists } \gamma = \mathrm{mgu}(\tilde{B}, \tilde{H}') \tag{2}$$

where $\tilde{H}'$ is the conjunction of the heads of the clauses H_i': $-\tilde{L}_i'$ and

$$\text{there exists a renaming } \psi \text{ such that } \gamma\psi = \rho\vartheta. \tag{3}$$

By definition of Γ_P, $c' = A\gamma$: $-\tilde{L}'\gamma \in \Gamma_P(J) \cap \mathscr{C}^{\Omega}$, where $\tilde{L}'$ denotes the conjunction obtained by replacing each atom B_i in $\tilde{B}$ by $\tilde{L}'_i$.

We prove now that $c' \leqslant_c c$. We have the following four steps.

(a) c' subsumes c.

For any i, c_i' subsumes c_i, since $c_i' \leqslant_c c_i$. Then, since the clauses are renamed apart, there exists a substitution σ such that $H_i'\sigma = H_i$ and $\mathrm{Set}(\tilde{L}_i'\sigma) \subseteq \mathrm{Set}(\tilde{L}_i)$. Without loss of generality we can assume that $\mathrm{Dom}(\sigma) \cap \mathrm{Var}(\bar{c}) = \varnothing$.

By Lemma A.7, $\gamma \leqslant \sigma\vartheta$. Then there exists a substitution α such that $\gamma\alpha = \sigma\vartheta$ and therefore $A\gamma\alpha = A\sigma\vartheta = A\vartheta$ (the last equality holds since $\mathrm{Dom}(\sigma) \cap \mathrm{Var}(A) = \varnothing$). Moreover, we have

$$\mathrm{Set}(\tilde{L}'\gamma\alpha) = \mathrm{Set}(\tilde{L}'\sigma\vartheta) \subseteq \mathrm{Set}(\tilde{L}\vartheta)$$

and the thesis holds.

(b) $A\gamma\psi = A\vartheta$.

By (3), $A\gamma\psi = A\rho\vartheta$ and therefore, since $\mathrm{Dom}(\rho) \cap \mathrm{Var}(\bar{c}) = \varnothing$, we have that $A\gamma\psi = A\vartheta$.

(c) $\mathrm{Rel}(c) \subseteq \mathrm{Rel}(c'\psi)$.

Let A be an atom in the body of c. By definition of relevant atom, $A \in \mathrm{Rel}(c)$ iff there exists $1 \leqslant i \leqslant n$ such that $B_i \in \mathrm{Rel}(\bar{c})$ and $A \in \mathrm{Rel}(c_i\vartheta)$. Moreover, $\mathrm{Rel}(c_i\vartheta) \subseteq \mathrm{Rel}(c_i'\rho\vartheta)$, since $c_i' \leqslant_c c_i$, and therefore, by (3), $A \in \mathrm{Rel}(c_i'\gamma\psi)$. Since ψ is a renaming, $B_i\psi \in \mathrm{Rel}(\bar{c}\psi)$ and then $A \in \mathrm{Rel}(c'\psi)$.

(d) $\mathrm{Set}(\mathrm{Rel}(c)) = \mathrm{Set}(\mathrm{Rel}(c'\psi))$.

This is analogous to (c). ∎

LEMMA A.8. *Let P, Q be programs such that* $\mathrm{Pred}(P) \cap \mathrm{Pred}(Q) \subseteq \Omega$. *Let us denote, for any $n \geqslant 0$, $T^{ca}_{P\cup Q}\uparrow n$ by I_n and $I_n \cap \mathscr{A}^{\Omega}$ by W_n. Then $T^{ca}_P(I_n) \subseteq T^{ca}_{P\cup W_n}\uparrow n+1$.*

Proof. The proof is by induction on n.

$(n=0)$

$$T^{ca}_P(I_0) = T^{ca}_P(\varnothing) = T^{ca}_P \uparrow 1$$

$(n>0)$.

First note that from the definitions we derive

$$T^{ca}_Q(I_{n-1}) \cap \mathscr{A}^{\mathrm{Pred}(P)} \subseteq T^{ca}_Q(I_{n-1}) \cap \mathscr{A}^{\Omega} \subseteq W_n. \tag{4}$$

We have the following relations:

$$\begin{aligned}
T^{ca}_P(I_n) &= T^{ca}_P(T^{ca}_{P\cup Q}(I_{n-1})) \\
&\quad \text{(by definition of } I_n\text{)} \\
&= T^{ca}_P(T^{ca}_P(I_{n-1}) \cup T^{ca}_Q(I_{n-1})) \\
&\quad \text{(by definition of } \cup\text{)} \\
&\subseteq T^{ca}_P(T^{ca}_{P\cup W_{n-1}}\uparrow n \cup T^{ca}_Q(I_{n-1})) \\
&\quad \text{(by inductive hypothesis)} \\
&\subseteq T^{ca}_P(T^{ca}_{P\cup W_n}\uparrow n \cup T^{ca}_Q(I_{n-1})) \\
&\quad \text{(since } W_{n-1} \subseteq W_n\text{)} \\
&\subseteq T^{ca}_P(T^{ca}_{P\cup W_n}\uparrow n \cup W_n) \\
&\quad \text{(by (4))} \\
&= T^{ca}_P(T^{ca}_{P\cup W_n}\uparrow n) \\
&\quad \text{(since } W_n \text{ contains only facts)} \\
&\subseteq T^{ca}_{P\cup W_n}(T^{ca}_{P\cup W_n}\uparrow n) \\
&\quad \text{(by definition of } T^{ca}_P\text{)} \\
&= (T^{ca}_{P\cup W_n})\uparrow n+1 \\
&\quad \text{(by definition of } \uparrow\text{)}
\end{aligned}$$

and the thesis holds. ∎

THEOREM 5.15. *Let P_1, P_2 be programs. Then $P_1 \not\approx_{(\Omega,\mathrm{ca})} P_2$ iff there exists a set of atoms $Q \subseteq \mathscr{A}^{\Omega}$ such that $P_1 \cup Q \not\approx_{\mathrm{ca}} P_2 \cup Q$.*

Proof. ($\Leftarrow$) This is straightforward by definition of $\approx_{(\Omega,\mathrm{ca})}$ and $\approx_{\mathrm{ca}}$.

($\Rightarrow$) If $P_1 \not\approx_{(\Omega,\mathrm{ca})} P_2$ then, by definition of $\approx_{(\Omega,\mathrm{ca})}$, there exists a program Y, such that $P_i \cup_{\Omega} Y$ is defined, for $i = 1, 2$ and $P_1 \cup_{\Omega} Y \not\approx_{\mathrm{ca}} P_2 \cup_{\Omega} Y$. Let $S_i = T^{ca}_{P_i\cup Y}\uparrow\omega$ for $i = 1, 2$. By Theorems 3.11 and 4.6, $S_1 \neq S_2$. Let $B = (S_1 \backslash S_2) \cup (S_2 \backslash S_1) \neq \varnothing$ and let $A \in B$ such that $A \in T^{ca}_{P_j\cup Y}\uparrow\bar{n}$, $j \in \{1, 2\}$, and $\bar{n}$ is the least n such that $(T^{ca}_{P_j\cup Y}\uparrow n) \cap B \neq \varnothing$ for $j = 1, 2$. Assume, without loss of generality, $j = 1$ and let $Q = (T^{ca}_{P_1\cup Y}\uparrow\bar{n}-1) \cap \mathscr{A}^{\Omega}$. By the definition of $\bar{n}$, $Q \subseteq S_2$. Therefore $T^{ca}_{P_2\cup Y}\uparrow\omega \supseteq T^{ca}_{P_2\cup Q}\uparrow\omega$ and therefore $A \notin T^{ca}_{P_2\cup Q}\uparrow\omega$. By Lemma A.8, $A \in T^{ca}_{P_1\cup Q}\uparrow\omega$. Since we have shown that $T^{ca}_{P_1\cup Q}\uparrow\omega \neq T^{ca}_{P_2\cup Q}\uparrow\omega$, the thesis holds by Theorems 3.11 and 4.6. ∎

LEMMA 5.16. *Let P be a program and Q be a set of atoms. Then $T^{ca}_{P\cup Q}\uparrow\omega = (T^{ca}_P + \mathrm{id})^{\omega}(Q)$.*

Proof. By definition of $\uparrow$, $T^{ca}_{P\cup Q}\uparrow\omega = \bigcup_{n<\omega} T^{ca}_{P\cup Q}\uparrow n$ and $(T^{ca}_P + \mathrm{id})^{\omega}(Q) = \bigcup_{n<\omega}(T^{ca}_P + \mathrm{id})^n(Q)$. We will then prove that $\forall n\ \exists m$ such that $T^{ca}_{P\cup Q}\uparrow n \subseteq (T^{ca}_P + \mathrm{id})^m(Q)$ and vice versa. The proof of the first part is by induction on n.

$(n=1)$ We have the following inclusions

$$\begin{aligned}
&T^{ca}_{P\cup Q}(\varnothing)\\
&\quad = T^{ca}_P(\varnothing)\cup T^{ca}_Q(\varnothing) && \text{(by Definition 3.8)}\\
&\quad = T^{ca}_P(\varnothing)\cup Q && \text{(since } Q \text{ is a set of atoms)}\\
&\quad \subseteq T^{ca}_P(Q)\cup Q && \text{(by monotonicity)}\\
&\quad = (T^{ca}_P + \mathrm{id})(Q) && \text{(by definition of + and id)}
\end{aligned}$$

$(n>1)$ Let us suppose by inductive hypothesis that $\exists m$ such that $T^{ca}_{P\cup Q}\uparrow n \subseteq (T^{ca}_P + \mathrm{id})^m(Q)$. Then

$$\begin{aligned}
&T^{ca}_{P\cup Q}\uparrow n+1\\
&\quad = T^{ca}_{P\cup Q}(T^{ca}_{P\cup Q}\uparrow n)\\
&\qquad \text{(by definition of }\uparrow)\\
&\quad \subseteq T^{ca}_{P\cup Q}((T^{ca}_P + \mathrm{id})^m(Q))\\
&\qquad \text{(by monotonicity and inductive hypothesis)}\\
&\quad \subseteq T^{ca}_P((T^{ca}_P + \mathrm{id})^m(Q)) \cup T^{ca}_Q((T^{ca}_P + \mathrm{id})^m(Q))\\
&\qquad \text{(by Definition 3.8)}\\
&\quad = T^{ca}_P((T^{ca}_P + \mathrm{id})^m(Q)) \cup Q\\
&\qquad \text{(by definition of } Q)\\
&\quad \subseteq (T^{ca}_P + \mathrm{id})((T^{ca}_P + \mathrm{id})^m(Q)) \cup Q\\
&\qquad \text{(by definition of +)}\\
&\quad = (T^{ca}_P + \mathrm{id})^{m+1}(Q)\\
&\qquad \text{(by definition of id and } (\ )^{m+1}).
\end{aligned}$$

To prove the converse we need the following fact which derives in a straightforward way from the monotonicity of T^{ca}_P and from the definitions:

$$T^{ca}_P(Q)\cup Q \subseteq T^{ca}_{P\cup Q}\uparrow 2. \tag{5}$$

The proof is now by induction on m.

$(m=1)$

$$\begin{aligned}
(T^{ca}_P + \mathrm{id})(Q) &= T^{ca}_P(Q)\cup Q && \text{(by definition of + and id)}\\
&\subseteq (T^{ca}_{P\cup Q})\uparrow 2 && \text{(by (5)).}
\end{aligned}$$

$(m>1)$ Let us suppose by inductive hypothesis that $\exists n$ such that $T^{ca}_{P\cup Q}\uparrow n \supseteq (T^{ca}_P + \mathrm{id})^m(Q)$. Then

$$\begin{aligned}
&(T^{ca}_P + \mathrm{id})^{m+1}(Q)\\
&\quad = (T^{ca}_P + \mathrm{id})((T^{ca}_P + \mathrm{id})^m(Q))\\
&\qquad \text{(by definition of } (\ )^{m+1})\\
&\quad \subseteq (T^{ca}_P + \mathrm{id})(T^{ca}_{P\cup Q}\uparrow n)\\
&\qquad \text{(by inductive hypothesis and monotonicity)}\\
&\quad = T^{ca}_P(T^{ca}_{P\cup Q}\uparrow n)\cup T^{ca}_{P\cup Q}\uparrow n\\
&\qquad \text{(by definition of id and +)}\\
&\quad \subseteq T^{ca}_{P\cup Q}(T^{ca}_{P\cup Q}\uparrow n)\cup T^{ca}_{P\cup Q}\uparrow n\\
&\qquad \text{(by Definition 3.8)}\\
&\quad = T^{ca}_{P\cup Q}\uparrow n+1 \cup T^{ca}_{P\cup Q}\uparrow n\\
&\qquad \text{(by definition of }\uparrow)\\
&\quad = T^{ca}_{P\cup Q}\uparrow n+1\\
&\qquad \text{(by monotonicity)}
\end{aligned}$$

and this completes the proof. ∎

PROPOSITION 5.22. *Let $I, J \subseteq \mathscr{C}$. The following properties hold:*

(i) $\sim_{(\Omega,s)}$ *is a congruence wrt infinite unions,*

(ii) *if* $I \sim_{(\Omega,s)} J$ *then* $\Gamma_P(I) \sim_{(\Omega,s)} \Gamma_P(J)$.

Proof. To prove (i) it is sufficient to observe that, given a set of integers N, if $I_n \sim_{(\Omega,s)} J_n$ holds for any $I_n, J_n \subseteq \mathscr{C}$ with $n \in N$, then

$$\begin{aligned}
&\left(\bigcup_{n\in N} I_n\right)\cap \mathscr{C}^{\Omega}\\
&\quad = \bigcup_{n\in N}(I_n \cap \mathscr{C}^{\Omega}) && \text{(by set theory)}
\end{aligned}$$

weakly subsumption equivalent to

$$\begin{aligned}
&\bigcup_{n\in N}(J_n \cap \mathscr{C}^{\Omega}) && \text{(by hypothesis)}\\
&\quad = \left(\bigcup_{n\in N} J_n\right)\cap \mathscr{C}^{\Omega} && \text{(by set theory).}
\end{aligned}$$

To prove (ii) we have to show that if c is a clause in $\Gamma_P(I)\cap\mathscr{C}^{\Omega}$ such that c is not a tautology, then there exists a clause c' in $\Gamma_P(J)\cap\mathscr{C}^{\Omega}$ such that c' subsumes c. Let us define $S_I = (I\cap\mathscr{C}^{\Omega})\cup \mathrm{Id}_{\Omega}$ and $S_J = (J\cap\mathscr{C}^{\Omega})\cup \mathrm{Id}_{\Omega}$.

Assume that $c \in \Gamma_P(I)\cap\mathscr{C}^{\Omega}$. By definition of Γ_P, there exists a clause $A: -\tilde{B} \in P$ and there exist n renamed apart clauses $c_i: H_i: -\tilde{L}_i$ in S_I, such that there exists $\vartheta = \mathrm{mgu}(\tilde{B}, \tilde{H})$ and c is the clause $(A: -\tilde{L})\vartheta$, where $\tilde{H}$ is the sequence of the heads of the n clauses c_i and $\tilde{L}$ is obtained by replacing each B_i in $\tilde{B}$ by $\tilde{L}_i$.

Since $I \sim_{(\Omega,s)} J$, S_I is subsumption equivalent to S_J. Then there exist n renamed apart clauses $c'_i: H'_i: -\tilde{L}'_i$ in S_J such

that c'_i subsumes c_i; i.e., there exists σ_i such that $H'_i\sigma_i = H_i$ and $\text{Set}(\tilde{L}'_i\sigma_i) \subseteq \text{Set}(\tilde{L}_i)$. Since the clauses are renamed apart, there exists a substitution σ such that $\sigma_{|\text{Var}(c'_i)} = \sigma_{i|\text{Var}(c'_i)}$. Now the proof is the same as that of (i) in Proposition 5.9. ∎

PROPOSITION 6.3. *Let $I, J \subseteq \mathscr{C}$. The following properties hold*:

(i) $\sim_{\text{pa}}$ *is a congruence wrt infinite unions*,

(ii) *if* $I \sim_{\text{pa}} J$ *then* $\Gamma_P(I) \sim_{\text{pa}} \Gamma_P(J)$.

Proof. The proof of (i) is straightforward by definition of $\sim_{\text{pa}}$.

To prove (ii) we have to show that if c: H: $-\tilde{B}$ is a clause in $\Gamma_P(I) \cup \text{Id}_H$, then there exists a clause c': K: $-\tilde{C}$ in $\Gamma_P(J) \cup \text{Id}_H$, such that $H = K\rho$ for a renaming ρ. Let us define $S_I = I \cup \text{Id}_H$ and $S_J = J \cup \text{Id}_H$.

If $c \in \text{Id}_H$ then the thesis is straightforward. Assume that $c \in \Gamma_P(I)$. By definition of Γ_P, there exists a clause $\bar{c}$: A: $-\tilde{B} \in P$ and there exist n renamed apart clauses c_i: H_i: $-\tilde{L}_i$ in S_I, such that

$$\text{there exists } \vartheta = \text{mgu}(\tilde{B}, \tilde{H}) \qquad (1)$$

and c is the clause $(A$: $-\tilde{L})\,\vartheta$, where $\tilde{H}$ is the sequence of the heads of the n clauses c_i and $\tilde{L}$ is obtained by replacing each B_i in $\tilde{B}$ by $\tilde{L}_i$.

The hypothesis $I \sim_{\text{pa}} J$ implies $S_I \sim_{\text{pa}} S_J$.

Then, by definition of $\sim_{\text{pa}}$, there exist n renamed apart clauses c'_i: H'_i: $-\tilde{L}'_i$ in S_J and there exists a renaming ψ_i such that $H'_i\psi_i = H_i$. Since the clauses are renamed apart, there exists a renaming ψ such that for any i, $\psi_{|\text{Var}(H'_i)} = \psi_{i|\text{Var}(H'_i)}$ and $\text{Dom}(\psi) \cap \text{Var}(\bar{c}) = \varnothing$. Therefore, by Lemma A.7 and by (1), there exists $\gamma = \text{mgu}(\tilde{B}, \tilde{H}')$, where $\tilde{H}'$ is the sequence of the heads of the clauses c'_i. Moreover,

$$\text{there exists a renaming } \rho \text{ such that } \psi\vartheta = \gamma\rho \qquad (2)$$

and, by definition of Γ_P, $(A$: $-\tilde{L}')\,\gamma \in \Gamma_P(J)$, with $\tilde{L}'$ obtained by replacing each B_i in $\tilde{B}$ by $\tilde{L}'_i$. Now it is sufficient to observe that

$$\begin{aligned} A\vartheta &= A\psi\vartheta \qquad (\text{since } \text{Dom}(\psi) \cap \text{Var}(\bar{c}) = \varnothing) \\ &= A\gamma\rho \qquad (\text{by } (2)) \end{aligned}$$

and this completes the proof. ∎

PROPOSITION 6.10. *Let $I, J \subseteq \mathscr{C}$. The following properties hold*:

(i) $\sim_{\text{pt}}$ *is a congruence wrt infinite unions*,

(ii) *if* $I \sim_{\text{pt}} J$ *then* $\Gamma_P(I) \sim_{\text{pt}} \Gamma_P(J)$.

Proof. The proof of (i) is straightforward by definition of $\sim_{\text{pt}}$.

To prove (ii) we have to show that H is a unit clause in $\Gamma_P(I)$ then there exists a clause K: $-\tilde{C}$ in $\Gamma_P(J)$ and there exists a renaming ρ such that $H = K\rho$.

Moreover, if H: $-B_1, ..., B_n$ is a (non-unit) clause in $\Gamma_P(I)$, then for $i = 1, ..., n$ there exists a clause K: $-C_1, ..., C_m$ in $\Gamma_P(J)$ such that H: $-B_i = (K$: $-C_j)\,\rho_i$ for a renaming ρ_i and $1 \leqslant j \leqslant m$.

Assume that $c \in \Gamma_P(I)$. By definition of Γ_P, there exists a clause $\bar{c}$: A: $-B_1, ..., B_n \in P$ and for $i = 1, ..., n$, there exists c_i: H_i: $-\tilde{L}_i$, renamed apart clause in $I \cup \text{Id}_H$, such that

$$\text{there exists } \vartheta = \text{mgu}((B_1, ..., B_n), (H_1, ..., H_n)) \qquad (1)$$

and c: $(A$: $-\tilde{L}_1, ..., \tilde{L}_n)\,\vartheta$.

Since $I \sim_{\text{pt}} J$, we have that $I \cup \text{Id}_H \sim_{\text{pt}} J \cup \text{Id}_H$. Assume now $\{|\tilde{L}_1, ..., \tilde{L}_n|\} \neq \{|\ |\}$ and let $B\vartheta$ be an atom in the body of c. Then there exists $1 \leqslant k \leqslant n$ such that $B\vartheta \in \{|\tilde{L}_k\vartheta|\}$. By definition of $\sim_{\text{pt}}$, for $i = 1, ..., n$, $i \neq k$, there exist a renamed apart clause c'_i: H'_i: $-\tilde{L}'_i$ in $J \cup \text{Id}_H$ and a renaming ψ_i such that $H_i = H'_i\psi_i$. Moreover, for $i = k$, there exists a renamed apart clause c'_k: H'_h: $-D_1, ..., D_m$ in $J \cup \text{Id}_H$ such that H_k: $-B = (H'_k$: $-D_l)\,\psi_k$ for a renaming ψ_k and $1 \leqslant l \leqslant m$.

Since clauses are renamed apart, there exists a renaming ψ such that, for $i = 1, ..., n$, $\psi_{|\text{Var}(c'_i)} = \psi_{i|\text{Var}(c'_i)}$ and $\text{Dom}(\psi) \cap \text{Var}(\bar{c}) = \varnothing$. Therefore, by Lemma A.7 and by (1), there exists $\gamma = \text{mgu}((B_1, ..., B_n), (H'_1, ..., H'_n))$. Moreover,

$$\text{there exists a renaming } \rho \text{ such that } \psi\vartheta = \gamma\rho \qquad (2)$$

and, by definition of Γ_P, $(A$: $-\tilde{L}'_1, ..., \tilde{L}'_{k-1}, D_1, ..., D_m, \tilde{L}'_{k+1}, ..., \tilde{L}'_n)\,\gamma \in \Gamma_P(J)$.

Now it is sufficient to observe that

$$\begin{aligned} (A\text{: } -B)\,\vartheta &= (A\text{: } -D_l)\,\psi\vartheta \\ &\qquad (\text{since } B = D_l\psi \text{ and } \text{Dom}(\psi) \cap \text{Var}(c) = \varnothing) \\ &= (A\text{: } -D_l)\,\gamma\rho \qquad (\text{by } (2)) \end{aligned}$$

and this completes the proof of the case $\{|\tilde{L}_1, ..., \tilde{L}_n|\} \neq \{|\ |\}$. The proof for the case $\{|\tilde{L}_1, ..., \tilde{L}_n|\} = \{|\ |\}$ is analogous to the previous one. ∎

ACKNOWLEDGMENTS

We thank the referees for their very useful comments.

Received November 10, 1992; final manuscript received September 22, 1994

REFERENCES

[1] Apt, K. R. (1990), Introduction to Logic Programming, in "Handbook of Theoretical Computer Science," Vol. B, "Formal Models

and Semantics" (J. van Leeuwen, Ed.), pp. 495–574. Elsevier, Amsterdam/MIT Press, Cambridge.

[2] Barbuti, R., Codish, M., Giacobazzi, R., and Maher, M. (1992), Oracle Semantics for PROLOG, *in* "Algebraic and Logic Programming, Proceedings of the Third International Conference" (H. Kirchner and G. Levi, Eds.), Vol. 632, pp. 100–114. Springer-Verlag, Berlin.

[3] Barbuti, R., Giacobazzi, R., and Levi, G. (1993), A general framework for semantics-based bottom-up abstract interpretation of logic programs, *ACM Trans. Programming Languages and Systems* **15**(1), 133–181.

[4] Bossi, A., Bugliesi, M., and Fabris, M. (1993), Fixpoint semantics for PROLOG, *in* "Proc. Tenth International Conference on Logic Programming" (D. S. Warren, Ed.), pp. 374–389. MIT Press, Cambridge, MA.

[5] Bossi, A., Bugliesi, M., Gabbrielli, M., Levi, G., and Meo, M. C. (1993), Differential logic programming, *in* "Proc. Twentieth Annual ACM Symposium on Principles of Programming Languages," pp. 359–370. ACM, New York.

[6] Bossi, A., and Cocco, N. (1993), Basic transformation operations for logic programs which preserve computed answer substitutions of logic programs, *J. Logic Programming* **16**, 47–87.

[7] Bossi, A., Gabbrielli, M., Levi, G., and Martelli, M. (1994), The *s*-semantics approach: Theory and applications, *J. Logic Programming* **19/20**, 149–197.

[8] Bossi, A., Gabbrielli, M., Levi, G., and Meo, M. C. (1994), A compositional semantics for logic programs, *Theoret. Computer Sci.* **122**(1/2), 3–47.

[9] Bossi, A., and Menegus, M. (1991), Una semantica composizionale per programmi logici aperti, *in* "Proc. Sixth Italian Conference on Logic Programming" (P. Asirelli, Ed.), pp. 95–109.

[10] Clark, K. L. (1979), "Predicate Logic as a Computational Formalism," Res. Rep. DOC 79/59, Dept. of Computing, Imperial College, London.

[11] Codish, M., Dams, D., and Yardeni, E. (1990), "Bottom-Up Abstract Interpretation of Logic Programs," Tech. Rep., Dept. of Computer Science, The Weizmann Institute, Rehovot; (1994) *Theoret. Computer Sci.* **124** (1), 93–126.

[12] Cousot, P., and Cousot, R. (1977), Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, *in* "Proceedings Fourth ACM Symposium Principles of Programming Languages," pp. 238–252. ACM, New York.

[13] Debray, S. K. (1994), Formal bases for dataflow analysis of logic programs, *in* "Advances in Logic Programming Theory" (G. Levi, Ed.), Oxford Univ. Press, London, pp. 115–182.

[14] Denis, F., and Delahaye, J.-P. (1991), Unfolding, procedural and fixpoint semantics of logic programs, *in* "STACS 91" (C. Choffrut and M. Jantzen, Eds.), Lecture Notes in Computer Science, Vol. 480, pp. 511–522. Springer-Verlag, Berlin.

[15] Eder, E. (1985), Properties of substitutions and unifications, *J. Symbolic Comput.* **1**, 1–46.

[16] Falaschi, M., and Levi, G. (1990), Finite failures and partial computations in concurrent logic languages, *Theoret. Computer Sci.* **75**, 45–66.

[17] Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. (1989), Declarative modeling of the operational behavior of logic languages, *Theoret. Computer Sci.* **69**(3), 289–318.

[18] Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. (1993), A model-theoretic reconstruction of the operational semantics of logic programs, *Inform. and Comput.* **102**(1), 86–113.

[19] Gabbrielli, M., Dore, G. M., and Levi, G., Observable semantics for constraint logic programs, *J. Logic Comput.* 133–171.

[20] Gabbrielli, M., and Levi, G. (1991), Modeling answer constraints in constraint logic programs, *in* "Proc. Eight International Conference on Logic Programming" (K. Furukawa, Ed.), pp. 238–252. MIT Press, Cambridge, MA.

[21] Gabbrielli, M., and Levi, G. (1991), On the semantics of logic programs, *in* "Automata, Languages, and Programming, 18th International Colloquium," (J. Leach Albert, B. Monien, and M. Rodriguez-Artalejo, Eds.), Lecture Notes in Computer Science, Vol. 510, pp. 1–19. Springer-Verlag, Berlin.

[22] Gabbrielli, M., Levi, G., and Meo, M. C. (1992), Observational equivalences for logic programs, *in* "Proc. Joint International Conference and Symposium on Logic Programming" (K. Apt, Ed.), pp. 131–145. MIT Press, Cambridge, MA.

[23] Gaifman, H., and Shapiro, E. (1989), Fully abstract compositional semantics for logic programs, *in* "Proc. Sixteenth Annual ACM Symposium on Principles of Programming Languages," pp. 134–142. ACM, New York.

[24] Gaifman, H., and Shapiro, E. (1989), Proof theory and semantics of logic programs, *in* "Proc. Fourth IEEE Symposium on Logic in Computer Science," pp. 50–62. IEEE Computer Soc. Press, New York.

[25] Kawamura, T., and Kanamori, T. (1988), Preservation of stronger equivalence in unfold/fold logic programming transformation, *in* "Proc. International Conference on Fifth Generation Computer Systems," pp. 413–422. Institute for New Generation Computer Technology, Tokyo.

[26] Lassez, J.-L., and Maher, M. J. (1984), Closures and fairness in the semantics of programming logic, *Theoret. Comput. Sci.* **29**, 167–184.

[27] Lassez, J.-L., Maher, M. J., and Marriott, K. (1988), Unification revisited, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 587–625. Kaufmann, Los Altos, CA.

[28] Lloyd, J. W. (1987), "Foundations of Logic Programming," 2nd ed. Springer-Verlag, Berlin.

[29] Lloyd, J. W., and Shepherdson, J. C. (1991), Partial evaluation in logic programming, *J. Logic Programming* **11**, 217–242.

[30] Maher, M. J. (1988), Equivalences of logic programs, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 627–658. Kaufmann, Los Altos, CA.

[31] Mancarella, P., and Pedreschi, D. (1988), An algebra of logic programs, *in* "Proc. Fifth International Conference on Logic Programming" (R. A. Kowalski and K. A. Bowen, Ed.), pp. 1006–1023. MIT Press, Cambridge, MA.

[32] Marriott, K., and Søndergaard, H. (1989), Semantics-based dataflow analysis of logic programs, *in* "Information Processing 89" (G. Ritter, Ed.), North-Holland, Amsterdam.

[33] O'Keefe, R. A. (1985), Towards an algebra for constructing logic programs, *in* "Proc. IEEE Symposium on Logic Programming," pp. 152–160. IEEE, New York.

[34] Palamidessi, C. (1990), Algebraic properties of idempotent substitutions, *in* "Proc. of the 17th International Colloquium on Automata, Languages and Programming" (M. S. Paterson, Ed.), Lecture Notes in Computer Science, Vol. 443, pp. 386–399. Springer-Verlag, Berlin.

[35] Turi, D. (1991), Extending *S*-models to logic programs with negation, *in* "Proc. Eighth International Conference on Logic Programming" (K. Furukawa, Ed.), pp. 397–411. MIT Press, Cambridge, MA.

[36] van Emden, M. H., and Kowalski, R. A. (1976), The semantics of predicate logic as a programming language, *J. ACM* **23**(4), 733–742.

[37] Wolfram, D. A., Maher, M. J., and Lassez, J.-L. (1984), A unified treatment of resolution strategies for logic programs, *in* "Proceedings Second International Conference on Logic Programming" (S.-Å. Tärnlund, Ed.), pp. 263–276.